



Scuola Internazionale Superiore di Studi Avanzati - Trieste

MATHEMATICS AREA
MATHEMATICAL ANALYSIS, MODELLING, AND APPLICATIONS

**MODELLING FLUID STRUCTURE
INTERACTION PROBLEMS USING
BOUNDARY ELEMENT METHOD**

Doctoral Dissertation of:
Nicola Giuliani

Supervisors:

Prof. Antonio DeSimone
Prof. Luca Heltai

PhD Coordinator for Mathematical Analysis, Modelling, and Applications:

Prof. Massimiliano Berti

Academic Year 2016-2017

SISSA - Via Bonomea 265 - 34136 TRIESTE - ITALY

Declaration

Il presente lavoro costituisce la tesi presentata da Nicola Giuliani, sotto la direzione dei Proff. Antonio DeSimone e Luca Heltai, al fine di ottenere l'attestato di ricerca post-universitaria Doctor Philosophiae presso la Scuola Internazionale Superiore di Studi Avanzati di Trieste (SISSA), Curriculum in Analisi Matematica Modelli e Applicazioni, Area di Matematica. Ai sensi dell'art. 1, comma 4, dello Statuto della SISSA pubblicato sulla G.U. no. 36 del 13.02.2012, il predetto attestato e' equipollente al titolo di Dottore di Ricerca in Matematica. Trieste, Anno Accademico 2016–2017.

*A Erio, Giuseppina,
Marisa e Renzo,
i miei nonni.*

Abstract

This dissertation investigates the application of Boundary Element Methods (BEM) to Fluid Structure Interaction (FSI) problems under three main different perspectives. This work is divided in three main parts: i) the derivation of BEM for the Laplace equation and its application to analyze ship-wave interaction problems, ii) the implementation of efficient and parallel BEM solvers addressing the newest challenges of High Performance Computing, iii) the developing of a BEM for the Stokes system and its application to study micro-swimmers.

First we develop a BEM for the Laplace equation and we apply it to predict ship-wave interactions making use of an innovative coupling with Finite Element Method stabilization techniques. As well known, the wave pattern around a body depends on the Froude number associated to the flow. Thus, we thoroughly investigate the robustness and accuracy of the developed methodology assessing the solution dependence on such parameter.

To improve the performance and tackle problems with higher number of unknowns, the BEM developed for the Laplace equation is parallelized using OpenSOURCE technique in a hybrid distributed-shared memory environment. We perform several tests to demonstrate both the accuracy and the performance of the parallel BEM developed. In addition, we explore two different possibilities to reduce the overall computational cost from $O(N^2)$ to $O(N)$. Firstly we couple the library with a Fast Multiple Method that allows us to reach for higher order of complexity and efficiency. Then we perform a preliminary study on the implementation of a parallel Non Uniform Fast Fourier Transform to be coupled with the newly developed algorithm Sparse Cardinal Sine Decomposition (SCSD).

Finally we consider the application of the BEM framework to a different kind of FSI problem represented by the Stokes flow of a liquid medium surrounding swimming micro-organisms. We maintain the parallel structure derived for the Laplace equation even in the Stokes setting. Our implementation is able to simulate both prokaryotic and eukaryotic organisms, matching literature and experimental benchmarks. We finally present a deep analysis of the importance of hydrodynamic interactions between the different parts of micro-swimmers in the prevision of optimal swimming conditions, focusing our attention on the study of flagellated “robotic” composite swimmers.

Acknowledgements

Such a journey wouldn't have been possible without the necessary freedom and guidance. I am therefore deeply grateful to my supervisors, Proff. Antonio DeSimone and Luca Heltai, for their mentoring and support, always ready when needed. I would like to deeply thank all the people working at mathLab laboratory, your insight and kindness have always deeply encouraged me.

I really thank Professor François Alouges and Doctor Matthieu Aussal from *CMAF - Centre de Mathématiques Appliquées - Ecole polytechnique* for the kindness, the support and the continuous insights during the three months in Paris studying fast convolution techniques.

I also thank Chiara, Lorenzo, Gianluca, Filippo, Carolina and all the friends and colleagues that have accompanied me and very often opened my mind in these four years, your constant support have lightened even the darker times. I would like to deeply thank Andrea who has been more than a colleague-roommate-friend during this period.

The last heartfelt thought is for my parents and their never-ending support, *Grazie di tutto*.

Contents

Introduction	1
1 Laplace Boundary Integral Formulation	7
1.1 Boundary Integral Formulation	8
1.2 Laplace Boundary Element Method	9
1.3 Results	14
1.3.1 Global refinement strategy	14
1.3.2 Local refinement strategy	18
1.3.3 High order basis functions	19
1.3.4 Complex geometries integration	21
1.3.5 Post process and gradient recovery	23
2 Linearized ship-wave interaction problems using Boundary Element Method and FEM-SUPG stabilization	25
2.1 Free surface boundary condition	27
2.1.1 Linearized boundary condition	28
2.1.2 Treatment of the linearized free surface condition	29
2.1.3 Full resolution technique	30
2.1.4 Streamwise Upwind Petrov Galerkin stabilization	31
2.2 Numerical validation	33
2.2.1 Submerged prolate spheroid	33
2.2.2 Wigley hull	39
3 Parallel implementation of a Boundary Element Method and its acceleration using a Fast Multipole Method	45
3.1 Parallel Boundary Element Method	47
3.1.1 Profiling	47
3.1.2 Domain decomposition and workbalance	48
3.1.3 BEM: strong scaling	49
3.1.4 BEM: weak scaling	51
3.2 Accelerated Boundary Element Method	52

Contents

3.2.1	Fast Multipole Algorithm	53
3.2.2	FMM validation	55
3.2.3	FMM hybrid parallelisation	56
3.2.4	FMM: strong scaling	60
4	Parallelization of type 3 Non Uniform FFT and its application in fast numerical convolution	63
4.1	NUFFT type 3	65
4.1.1	Mathematical background	65
4.1.2	Algorithm key-steps	65
4.2	Shared memory parallelism	66
4.2.1	Implementation	66
4.2.2	Strong scaling analysis up to 16 threads	69
4.3	Distributed memory parallelism	71
4.3.1	Index sets creation	71
4.3.2	Implementation	72
4.3.3	Strong scaling analysis up to 16 processors	72
4.3.4	MPI TBB Comparison	73
4.3.5	64 bits indices compatibility	75
4.4	Introducing new features in the BlackNUFFT framework	77
4.5	Application to fast numerical convolution: Sparse Cardinal Sine Decomposition	77
4.5.1	The Sparse Cardinal Sine Decomposition	78
4.5.2	Algorithm key-steps	81
4.5.3	Parallelization strategies	81
5	Stokes Boundary Integral Formulation	87
5.1	Boundary Integral Formulation	87
5.2	Stokes Boundary Element Method	90
5.3	Kernels for specific interfaces	91
5.3.1	Single perfect slip flat interface	92
5.3.2	Kernel approximation for two perfect slip interfaces	99
6	Micro-motility analysis using Boundary Element Method	101
6.1	Mathematical modeling of micro-swimmers	102
6.1.1	Kinematic model	103
6.1.2	Fluid model	104
6.1.3	Swimmer model	105
6.2	Numerical resolution of the swimming problem	107
6.3	Geometrical reconstruction of a micro-swimmer	109
6.4	Numerical validation	113
6.4.1	Calculation of resistance matrices	114
6.4.2	Towed sphere near a wall	116
6.4.3	Two spheres analysis	116
6.4.4	Composite model swimmer	118
6.5	Experimental application	126

6.6	Head-Tail interactions in a model “robotic” bacterium	128
6.6.1	Optimal linear velocity in additive approach	128
6.6.2	Additive vs Global approach for linear and angular velocity . . .	131
6.6.3	A simple formula provides a correction for the additive approach	132
6.6.4	Convergence of resistance coefficients	137
6.6.5	Correction of RFT predictions	139
6.6.6	Efficiency	140
Conclusions and further developments		143
Bibliography		145

Introduction

This dissertation is devoted to the study of Fluid Structure Interaction (FSI) problems, such kind of phenomena arises when a fluid interacts with a given solid structure. The design of many engineering systems, *e.g.* ships, aircrafts, engines and bridges, is deeply influenced by this kind of interactions. Neglecting such phenomena can lead to catastrophic results, one of the most notorious of them is the failure of the first Tacoma Narrows Bridge. Even in biology many complex phenomena as the blood flow through soft vessels or the swimming of micro-organism in water are straightforwardly modeled as a FSI problems.

Multi physics problems such as FSIs, are often extremely complex and their analytical resolution is impossible in almost all cases of practical interest. Such problems usually have to be analyzed by means of experiments or numerical simulations. The continuous developing of High Performance Computing (HPC) facilities and numerical techniques opens new perspective for very demanding simulations as the ones FSI usually requires. When compared to experiments numerical simulations offer several advantages, both from an economic and a scientific point of view. In fact, numerical simulations are often cheaper than experiments and they are easily reproducible and customizable. Moreover, especially when we look to micro-biology, numerical simulations can naturally treat extremely small settings while the building of a dedicated experimental facilities is extremely expensive, sometimes even impossible. For all these reasons the validation of the reliability of such numerical tools is of paramount importance. In this work, we present numerical solvers for FSI problems along with extensive validation procedures to verify both the accuracy and the efficiency of the developed methods.

In this dissertation the fluid component (water) is modeled as an incompressible Newtonian fluid and the incompressible Navier–Stokes (NS) equations are the governing equations of the corresponding mathematical model. The numerical resolution of the complete NS system involves non linear solvers and time integration schemes that often require a huge amount of computational resources. It is well known that the direct numerical simulation (DNS) of NS equations requires an accurate discretization of all the turbulent structures of the flow [64, 75, 117]. Turbulence models (as Reynolds Average Navier Stokes equations or Large Eddy Simulation) offer some alternative to DNS,

however they depend on the specific flow considered and they are still very costly from a computational point of view [37, 54, 112, 118]. Finite Element Method (FEM) and Finite Volume Method (FVM) are the most common numerical schemes for the discretization of hydrodynamic flow equations. Such numerical methodologies are quite general and allow for accurate resolution even when non linear unsteady problems are considered, as in the case with Navier–Stokes equations. FEMs and FVMs are based on the complete three dimensional simulation of the fluid problem, so an accurate geometrical mesh reconstruction of the entire fluid domain is mandatory. Whenever the domain undergoes severe deformations (a common situation in FSI problems) satisfying the latter requirement is far from trivial. For these reasons it is convenient, when possible, to simplify the mathematical setting modeling the flow, this can be done considering the peculiarities of the specific problem considered. In many cases, it is possible to simplify Navier–Stokes equations to retrieve simpler, more stable mathematical formulations, also sparing computational resources.

In particular, whenever the Partial Differential Equation (PDE) modeling the problem admits a fundamental solution, it can be recast as a Boundary Integral Equation (BIE) [113]. In such a way the unknowns on the boundary of the domain are expressed as functions of quantities living only on the boundary of the domain itself. Boundary Element Methods (BEM) arise as the numerical discretization of such BIEs. For BEM only a boundary mesh is required, which deeply mitigates the grid generation complexity especially in presence of large domain deformations.

In the present work, we tackle two different FSI problems making use of the BEM approach. The first problem is prediction of the wave drag of a body advancing in calm water and the second is the simulation of micro-swimmers. On one hand, the study of ship wave interactions is of capital importance in ship-design processes [22, 42, 78, 81, 98], on the other hand, the study micro-swimmer offers insights on industrial (bacteria pollution) and medical (spermatozoa, bacteria) processes. We see the motility of swimming cells as the study of the flow induced by the swimming motion of the swimmer [40, 91, 95, 109, 110]. For what concerns the structural problem we consider the position of the solid-fluid interface as a given Dirichlet datum for both cases. Quite surprisingly the two problems, which are obviously completely different, can be analyzed in a very similar way.

When we study ship-wave interactions, we look to the adimensionalization of Navier–Stokes equations and we see that the Reynolds number, expressing the relative importance of the inertial forces with respect to viscous stresses, is $> 10^7$). Therefore it is reasonable to assume that the viscous forces within the fluid are negligible with respect to inertial effects. Based on the assumption that the flow is irrotational the velocity of the fluid is expressed as the gradient of a scalar quantity called kinematic potential. In such a case, we rewrite the mass conservation as the standard Laplace equation and the linear momentum balance as the Bernoulli equation. The fundamental solution for the governing Laplace equation is well known, so a Boundary Integral Formulation for the problem is possible. BEMs for this kind of problems have been derived in many scientific fields, from aeronautics [34, 72, 82], to hydrodynamics [22, 29, 42, 81, 98, 120]. To completely neglect the viscous forces we must assure that no lift forces are involved, otherwise we need to account for the presence of a wake [82]. This is the typical setting of a ship advancing in calm water, where the viscous forces only induce a pure drag

force which can be estimated by means of empirical algebraic formulas [79]. Models based upon the potential flow theory are very often used in the early stages of ship design, as they are particularly suitable for a quick estimation of the drag component resulting from the wave generation of an advancing hull. The application of BEM to free surface flows is well documented in literature [23, 29, 77, 78, 98, 120].

Under the condition that only small waves are created (i.e., if we consider the wave amplitude A and its wavelength λ we have that $A \ll \lambda$), it is possible to apply a perturbation analysis to the fully nonlinear free surface boundary conditions, to obtain a linearized free surface boundary condition for the flow potential [55, 85]. The importance of such a linear condition lies in the fact that superposition principle can be applied to separately evaluate the flow fields related to diffraction and radiation. In addition, since the linearized condition is imposed on the undisturbed free surface, the computational grid does not need to be deformed during the simulations, leading to an enormous computational advantage with respect to the fully non-linear case. However, the approximation due to the linearization process lead to a symmetric condition in the stream wise direction possibly causing unphysical results in presence of a main stream [107]. It is well known that waves generated by the motion of a body propagate mostly downstream, originating the so called Kelvin wake pattern. Several methods have been developed for annihilation of forward propagating waves, which result in the correct solution of the linearized free surface potential problem.

Among others, in [84, 86] the authors studied the possibility to employ *ad hoc* Green functions which automatically satisfy the linearized condition and suppress unphysical waves propagating upstream. Such works led to the developing of the so-called Neumann-Kelvin (NK) methods. The highly oscillatory behavior of such Kelvin sources [24, 116], makes even more difficult the —already critical— evaluation of BIEs on the boundary of the domain. A different strategy, proposed by Dawson in [29], consists in suppressing the undesired upstream waves at the numerical level, by employing an upwind finite difference approximation of the second order derivatives of the potential which appear in the linearized free surface boundary conditions. Dawson’s method leads to accurate results, however it presents two main drawbacks: it requires the preliminary solution of an additional BEM problem (to align the grid with the streamwise direction), and the use of an upwind finite difference scheme requires the generation of a structured computational grid, where the nodes are aligned in the streamwise direction. It is generally more difficult to generate such grids with respect to unstructured meshes, which can adapt more successfully to complex geometric configurations. Moreover, the upwind finite difference scheme is not easily compatible with a local adaptive refinement strategy. We present a modification of Dawson’s approach that allows the use of unstructured non conformal grids. Streamline Upwind Petrov Galerkin (SUPG) provides a stabilization strategy [2] which suppresses unphysical wave patterns. The additional terms coming from the stabilization are independent of the grid employed, making it possible to use unstructured, and even non conformal grids. SUPG is a common stabilization method in the field of FEMs. Its application to the numerical solution of fully nonlinear potential free surface problems through BEM has been discussed for the first time in [77, 78]. We describe the application of SUPG to the linearized free surface potential model. The methodology proposed allows for the use of arbitrary order boundary elements, and is even compatible with isogeometric

discretizations [16]. We propose two test cases to validate our implementation: the first one consists of a fully submerged spheroid advancing at constant speed in calm water, the second one consists of a Wigley hull advancing at constant speed in calm water.

Micro-biology offers a completely different kind of FSI problems, represented by the study of micro-swimmers. While such problem and the ship-wave interaction are obviously extremely different, they can be modeled in a very similar way. Even when we address micro-biology problems we must consider the adimensionalization of Navier–Stokes equations, see [95]. It is well known that the different parts of the Navier–Stokes system scale differently with respect to the problem length scale, so we consider some simplification due to the actual problem dimension. If we evaluate the cell dimension to be $O(10^{-5})$ meters, and its typical velocity to be $O(10^{-5})m/sec$, we obtain that the Reynolds number is $O(10^{-4})$. This implies that viscous forces dominate the flow around a micro-swimmer (we are in the opposite scenario with respect to ship-wave interactions). In other terms, the flow around the cell can be modeled as a Stokes flow. Even for the Stokes system the fundamental solution is very well known, so it can be recast as a BIE. Swimmers exploit periodic shape change to achieve a net rigid motion, such shape change often involves large deformation of the computational domain, making the use of standard FEM far from trivial. For such reason, BEMs are often chosen as resolution strategy for this kind of problems, and the application of BEMs to micro-swimmers is very well documented in literature [90, 91, 97, 109]. The micro-swimming behavior of motile cells can give interest insights on many complex biological processes. In particular, the contamination induced by bacteria, or other pollutants, influences many different applications, both of industrial and physical interest. Many micro-organisms may change their swimming behavior depending on the fluid properties [70] or near particular interfaces [91, 109], deeply influencing the pollution or contamination of water reserves. In physics and medicine the study of motile pathogens or sperm cell is of great interest and can offer new insight in the treatment of several diseases [59] or in the understanding of different reproduction strategies [5]. From an experimental point of view, the prediction of the swimming behaviors allows for the optimization of the performance of artificial “robotic” swimmer [27, 88]. For these reasons we are interested in evaluating the performance associated with the motility strategies of micro-swimmers, both to study biological existing organisms and to predict the performance of “robotic” micro-swimmers under design.

In the last decades many methods have been developed to simplify the mathematical modeling of micro-swimmers even further. These theories are derived considering both the linearity and the locality of the Stokes system. The most significant example is given by the Resistive Force Theory (RFT) [44, 69], that provides a relationship between the force acting on a point of the swimmer and the local velocity of the point itself. In other cases, hydrodynamic interactions between different parts of composite swimmers are neglected. Additive approximations (AA) [66, 96] are very used, but they introduce an error due to the intrinsic non locality of the mathematical model [100]. We present a BEM methodology to study both prokaryotic and eukaryotic organism coupling different OpenSOURCE programs. We validate our methodology against several literature benchmarks [91, 100, 109], we present an application to an experimental setup [50], and we present an analysis of the hydrodynamic interactions between different bodies [96].

The numerical implementation of BEMs, both for the Laplace and the Stokes setting,

is challenging from many points of view. Boundary Integral Equations are characterized by convolution operators between fundamental solutions and the unknowns on the boundary. Such boundary integral operators are weakly singular when valued on the boundary and they need to be properly approximated by means of numerical quadrature formulas. We need to implement specific singular quadrature [114], to handle the presence of singularity on the boundary. Moreover, although the requirement of the discretization of the boundary alone leads to a clear reduction of the number of unknowns N of the fluid domain, the fundamental solutions are not compactly supported, resulting in dense solving systems. This implies a typical time to solution of at least order $O(N^2)$, this means that the computational cost of BEM systems increases quadratically with the number of unknowns. By this perspective, the continuous developing of computer architectures opens new possibilities for an efficient and high performing implementation of Boundary Element Methods. In this work we address such computational challenges exploring possible ways of exploiting recent parallel programming techniques to increase the computational efficiency of BEMs. In particular we focus on hybrid parallelization strategies coupling shared and distributed memory parallelisms: we use Intel Threading Building Blocks [99], as shared memory parallelism, and the standard Message Passing Interface as distributed memory paradigm. The derived hybrid parallelization strategy allows for an efficient computation on modern HPC platforms [63]. In recent years many methods have been developed to approximate the action of a BEM matrix-vector product in order $O(N)$ operations instead of assembling the full matrices. Among the other possible accelerators (among which we mention Hierarchical matrices [14, 15, 43, 58]) we couple the parallel BEM with a Fast Multiple Method (FMM) [46]. Parallel versions of the FMM algorithm have been derived in recent years [13, 45, 121] using shared and distributed memory paradigm. We present a BEM-FMM [41] coupling exploiting hybrid shared-distributed memory parallelization strategy that can naturally treat very general kind of Laplace BEM problems including mixed boundary conditions, high order elements, local refinement strategies and complex geometries (via direct integration with CAD data structures).

Recently, a very promising FFT based method called Sparse Cardinal Sine Decomposition (SCSD) [3] has been developed and it has shown very promising results (comparable to FMM). As far as the authors know, a parallel SCSD algorithm has not been developed yet. We investigate a possible SCSD parallelization strategy. The algorithm [3] is based on the Non Uniform Fast Fourier Transform (NUFFT) of type 3 [31, 47, 67]. The first step for the parallelization of SCSD is to retrieve a parallel NUFFT. Such a technique is an accelerator of the Discrete Fourier Transform (DFT) between arbitrary points in space and frequency. In fact, the standard Fast Fourier Transform (FFT) requires the points to be on an equi-spaced grid. The basic idea of NUFFTs is to interpolate the arbitrary distribution on points on a regular grid and to apply a FFT, or pruned FFT [92], to mitigate the natural quadratic cost of standard DFTs. Over the last decades some attempts have been made to obtain a parallel NUFFT [12, 92], however their implementation is very often hardware optimized and specific [89], and their usage is sometimes far from trivial. We present a novel parallel NUFFT library based on a standard FFT library (FFTW [33]) as backend FFT library. The developed library [39] is highly modular, OpenSOURCE (released under GPL license) and it can be straightforwardly extended by the scientific community. We

apply the developed NUFFT library to parallelize the complete SCSD algorithm. Even if this is a preliminary result we present an extensive performance analysis both for the NUFFT alone and the SCSD algorithm.

This dissertation is organized as follows, in Chapter 1 we introduce the mathematical formulation for the Laplace Boundary Integral Equation and the derivation of a Boundary Element Method, we present some tests to assess the accuracy of the resulting numerical method. In Chapter 2 we apply the BEM derived to the study of ship-wave fluid structure interaction problems using a stabilized linearized free surface boundary condition that allows for a straight forward implementation of high order elements with local refinement techniques. In Chapter 3 we develop a High Performance Computing library for the resolution of the Laplace BIE presented in Chapter 1. We use existing OpenSOURCE libraries to develop an efficient hybrid parallelization including a coupling with a Fast Multiple Method. In Chapter 4 we present a flexible and modular parallelization of the Non Uniform Fast Fourier Transform of type 3, using OpenSOURCE HPC libraries. We use the developed library to derive a preliminary parallel implementation of another BEM accelerator, the Sparse Cardinal Sine Decomposition. In Chapter 5 we introduce the mathematical setting leading to the boundary integral formulation of the Stokes System. We present possible Green functions satisfying different boundary conditions, and we describe the numerical implementation of a Boundary Element Method for the Stokes system. Finally in Chapter 6 we address the motility strategies of micro-swimmers. We describe the mathematical modeling of the problem and we present various results to prove both the flexibility and the accuracy of the present work. We present a deep analysis of hydrodynamic interactions in flagellated bacteria that highlights their importance in the determination of optimal swimming conditions.

CHAPTER 1

Laplace Boundary Integral Formulation

In the present Chapter we discuss the mathematical and numerical aspects of the Boundary Integral Formulation for the Laplace equation. Over the last decades the Laplace equation has often been solved using a boundary integral formulation, and indeed in many fields of engineering it is quite common to solve such equation through the Boundary Element Method (BEM). Natural fields of application are electromagnetism and thermal conductivity. Since a fundamental Green function is known, the problem can be recast in a Boundary Integral Equation involving only the boundary of the domain. This is what makes this formulation appealing for the present work. In fact, it is possible to solve the Laplace equation only discretizing the boundary of the domain defining the so-called Boundary Element Method.

A number of steps are possible to improve the applicability of BEM. We start by mentioning high order elements, which reduce the number of degrees freedom required to obtain a specific tolerance for problems with smooth solutions. High order BEMs are more attractive when compared to their finite element counterparts, since for finite elements, increasing the order results in a less sparse, more ill conditioned, system matrix. For BEMs the matrices are already full, and this is no longer a disadvantage. When non smooth problems are considered, high order elements can be combined with local refinement techniques to reduce the final size of the problem. In general these two techniques together work well when the domain itself is smooth. They are no longer sufficient when the domain is only Lipschitz: if the domain presents a sharp edge the normal vector has a jump across such interface, inducing a jump also in the normal component of the solution gradient. While this is not an issue for discontinuous elements of arbitrary orders, it may be an issue if we consider continuous elements. A possible solution is the so-called double nodes technique [48], where continuity is preserved only on the solution, while its normal gradient is allowed to have a jump

across physical edges. The usage of this technique allows for an accurate solution of mixed Neumann Dirichlet boundary value problems on domains with sharp edges. In recent years the developing of Computer Aided Design technologies has posed new challenges to both FEM [25] and BEM communities [56]. The integration of complex CAD geometries in BEM widens the range of the application [80]. We exploit refining and integration strategies directly on the exact geometry specified by user provided files.

The Chapter is organised as follows, in Section 1.1 we describes the Boundary Integral Formulation for the Laplace equation, in Section 1.2 we discuss the numerical implementation of a Boundary Element Method and in Section 1.3 we report and discuss some results of the developed BEM.

1.1 Boundary Integral Formulation

Following the same formalism in [19, 42], we consider a bounded open domain Ω with Lipschitz boundary $\Gamma = \partial\Omega$, and we solve the model problem

$$-\Delta\phi = 0 \quad \text{in } \Omega \quad (1.1a)$$

$$\frac{\partial\phi}{\partial n} = f_N(\mathbf{x}) \quad \text{on } \Gamma_N \quad (1.1b)$$

$$\phi = f_D(\mathbf{x}) \quad \text{on } \Gamma_D, \quad (1.1c)$$

where Dirichlet and Neumann boundary conditions are imposed on the portions Γ_D , and Γ_N of $\partial\Omega$, such that $\Gamma_D \cup \Gamma_N = \partial\Omega$, $\Gamma_D \cap \Gamma_N = \emptyset$, and $\Gamma_D \neq \emptyset$.

We can multiply equation (1.1a) by an arbitrary function G and integrate by parts twice to obtain the second Green identity

$$\int_{\Omega} (-\Delta\phi)G \, dx = \int_{\Omega} (-\Delta G)\phi \, dx + \int_{\Gamma} \frac{\partial\phi}{\partial \mathbf{n}} G \, ds - \int_{\Gamma} \phi \frac{\partial G}{\partial \mathbf{n}} \, ds = 0, \quad (1.2)$$

where \mathbf{n} is the outward normal to Γ . Choosing G to be the so called free-space Green function, or fundamental solution of the Laplace equation:

$$G(\mathbf{r}) := \frac{1}{2\pi} \ln \left(\frac{1}{|\mathbf{r}|} \right) \quad \text{if } \mathbf{r} \in \mathbb{R}^2, \quad (1.3a)$$

$$G(\mathbf{r}) := \frac{1}{4\pi|\mathbf{r}|} \quad \text{if } \mathbf{r} \in \mathbb{R}^3, \quad (1.3b)$$

we can exploit equation (1.2) and the defining properties of the Dirac delta distribution to express the boundary integral formulation of the Laplace equation:

$$\phi(\mathbf{x}) = \int_{\Gamma} G(\mathbf{x} - \mathbf{y}) \frac{\partial\phi}{\partial n}(\mathbf{y}) \, ds_y - \int_{\Gamma} \phi(\mathbf{y}) \frac{\partial G}{\partial n}(\mathbf{x} - \mathbf{y}) \, ds_y \quad \forall \mathbf{x} \in \Omega. \quad (1.4)$$

Equation (1.4) allows for the computation of the potential ϕ in any point \mathbf{x} in the domain Ω if $\phi(\mathbf{x})$ and its normal derivative $\frac{\partial\phi}{\partial n}(\mathbf{x})$ are known on the boundary Γ . If we consider the trace of the representation formula (1.4), the kernels $G(\mathbf{x} - \mathbf{y})$ and

$\frac{\partial G}{\partial n}(\mathbf{x} - \mathbf{y})$ become weakly singular (but integrable) and singular respectively. Considering the Cauchy Principal Value (CPV) of the singular integral, we can write the boundary integral form of the original problem as

$$\alpha(\mathbf{x})\phi(\mathbf{x}) = \int_{\Gamma} G(\mathbf{x} - \mathbf{y}) \frac{\partial \phi}{\partial n}(\mathbf{x}) \, ds_y - \int_{\Gamma}^{PV} \phi(\mathbf{x}) \frac{\partial G}{\partial n}(\mathbf{x} - \mathbf{y}) \, ds_y \quad \text{on } \Gamma \quad (1.5a)$$

$$\frac{\partial \phi}{\partial n} = f_N(\mathbf{x}) \quad \text{on } \Gamma_N \quad (1.5b)$$

$$\phi = f_D(\mathbf{x}) \quad \text{on } \Gamma_D, \quad (1.5c)$$

or, by explicitly writing the boundary conditions in the boundary integral equation,

$$\begin{aligned} & \chi_{\Gamma_N}(\mathbf{x})\alpha(\mathbf{x})\phi(\mathbf{x}) - \int_{\Gamma_D} G(\mathbf{x} - \mathbf{y}) \frac{\partial \phi}{\partial n}(\mathbf{x}) \, ds_y + \int_{\Gamma_N}^{PV} \phi(\mathbf{x}) \frac{\partial G}{\partial n}(\mathbf{x} - \mathbf{y}) \, ds_y \\ &= -\chi_{\Gamma_D}(\mathbf{x})\alpha(\mathbf{x})f_D(\mathbf{x}) + \int_{\Gamma_N} G(\mathbf{x} - \mathbf{y})f_N(\mathbf{x}) \, ds_y - \int_{\Gamma_D}^{PV} f_D(\mathbf{x}) \frac{\partial G}{\partial n}(\mathbf{x} - \mathbf{y}) \, ds_y. \end{aligned} \quad (1.6)$$

The coefficient $\alpha(\mathbf{x})$ is obtained from the CPV evaluation of the singular integral, and it represents the fraction of solid angle with which the domain Ω is seen from the boundary point \mathbf{x} . We use χ_A to indicate the characteristic function, i.e., $\chi_A(\mathbf{x})$ is one if $\mathbf{x} \in A$, and zero otherwise. Equation (1.6) is also known as Boundary Integral Equation (BIE). With such representation it is possible to derive the potential where its normal derivative is known, and viceversa.

1.2 Laplace Boundary Element Method

By taking the trace of the boundary integral representation (1.4), one obtains the integral formulation (1.6) which only lives on the boundary. If one is only interested in obtaining the solution of the problem on the boundary Γ , a numerical solution can be obtained through the discretization of the BIE (1.6). We introduce a suitable discretization of the approximation spaces and of the boundary, by defining standard Lagrangian finite element spaces on Γ as basis functions both for the geometry and the unknowns ϕ and $\frac{\partial \phi}{\partial n}$. We refer to this type of approximation as Isoparametric BEM. The unknowns of the problem are the values of ϕ and $\frac{\partial \phi}{\partial n}$ on the respective set of interpolatory points of the Finite Element discretization. In general, the finite dimensional approximation of the problem can be divided in 5 main steps:

Computational mesh generation We define a quadrilateral computational mesh meant as a regular decomposition Γ_h of the boundary Γ . Here *regular* means that any two cells K, K' only intersects on common faces, edges or vertices. In all the computations carried out in this work we provide a coarse grid and then we progressively refine it so as to obtain a solution with specified accuracy. From the user's perspective, this eases the generation of an initial grid. This requires that the solver is provided with a suitable description of the geometry on which the mesh must be refined. Along with some native shapes described by simple analytical formulas, such as spheres, toruses, cubes, pyramids, *etc.* several applications require the introduction of arbitrary geometrical descriptions which are commonly contained in CAD files. Any surface can be refined

on its own geometry, thus we offer the possibility to import a CAD geometry and use it for mesh refinement purposes. This feature has been employed, for instance, in ship-wave simulation through BEM, [79], and an aeronautics-like example of a NACA wing shape is presented in Section 1.3.4. As said, the framework described allows for a very minimal effort for the generation of the first numerical discretization, which can be further refined until the error in the simulation becomes acceptable. Of course such feature can be naturally and effectively exploited in combination with local refinement strategies. This feature, which is known to work nicely in the Finite Element framework, is based on the development of a proper *a posteriori* error estimator guiding the refinement of the mesh. Several works confirm that such strategy can be successfully applied to BEMs. For example in [60] the authors show an error analysis based on the construction of complementary spaces with respect to the current finite element spaces. Other possible estimators are based on Hierarchical spaces [21] or classical residual evaluation [32]. We present the results obtained with an estimator that is very simple to implement and has already proved effective in its applications to BEMs (see [42]). The estimator was introduced in [1] for volume problems, and generalized to surface problems in [30]. It works by approximating the error on each cell by considering the L_2 norm of the jump of the surface gradient of the approximated solution on each face.

Definition of the discrete spaces We define two finite dimensional spaces V_h and Q_h following [42],¹ defined on Γ_h , such that

$$V_h := \{\phi_h \in L^2(\Gamma_h) : \phi_{h|K} \in \mathbb{Q}^r(K), K \in \Gamma_h\} \equiv \text{span}\{\psi_i\}_{i=1}^{N_V} \quad (1.8a)$$

$$Q_h := \{\gamma_h \in L^2(\Gamma_h) : \gamma_{h|K} \in \mathbb{Q}^s(K), K \in \Gamma_h\} \equiv \text{span}\{\omega_i\}_{i=1}^{N_Q}, \quad (1.8b)$$

where on each cell K , located on the boundary, $\phi_{h|K}, \gamma_{h|K}$ are polynomial functions of degree r and s respectively, in each coordinate direction. In principle these two spaces can be built independently, but for the moment we restrict the discussion to the case where we use the same Finite Element discretization for both the primal and the dual unknown, i.e., $V_h = Q_h \equiv \text{span}\{\psi_i\}_{i=1}^{N_V}$. A preliminary study of such issue has been carried out in [38], where it was pointed out that the use of different spaces for ϕ and $\frac{\partial \phi}{\partial n}$ does not lead to a significant accuracy gain with respect to using the same basis functions for the two unknowns. We use a classical iso-parametric formulation, using the same space both for the geometry and for the unknowns. Since we are using geometries specified by CAD files an interesting choice would be using NURBS function at least for the geometry to guarantee its exact preservation. However, the treatment of arbitrarily complex CAD (usually defined by many different patches, possibly with trimmed surfaces) poses many implementation challenges that are currently being addressed. We reach a compromise by using iso-parametric discretization based on standard Q_N Lagrangian finite elements, and by collocating the support points of the geometry patches directly on the CAD surfaces. Such finite dimensional spaces allow an accurate and

¹ For the integrals in equation (1.5a) to be bounded, ϕ and $\frac{\partial \phi}{\partial n}$ must lie in the spaces V and Q , defined as

$$V := \left\{ \phi \in H^{\frac{1}{2}}(\Gamma) \right\} \quad (1.7a)$$

$$Q := \left\{ \gamma \in H^{-\frac{1}{2}}(\Gamma) \right\}, \quad (1.7b)$$

where $\Gamma = \partial\Omega$. We recall that $H^{\frac{1}{2}}(\Gamma)$ is usually defined as the space of traces on Γ of functions in $H^1(\Omega)$, while $H^{-\frac{1}{2}}(\Gamma)$ is its dual space. The spaces V_h and Q_h are constructed as conforming finite dimensional subspaces of V and Q respectively.

refined representation of both the geometry and the unknowns. Compatibility of the arbitrary order finite dimensional spaces with sharp edges in the geometry is obtained through the double nodes technique, [49]. In presence of a geometrically sharp edge, the normal vector is discontinuous, and so is the potential normal derivative. The use of double nodes allows for the presence of such discontinuity by a local enrichment of the space Q_h .

Collocation of the Boundary Integral Equations While the Galerkin method appears as the natural choice for FEMs in BEM framework this method implies a second integration of weakly singular kernels. We already treat in a specific way the singular integrals that naturally appears in the BIE using specific quadrature formulas. A second integration would increase the computational complexity of the BEM, and we therefore focus on collocation methods. In Section 1.3.1 and 1.3.3 we show both the accuracy of the presented collocation method and the criticality of proper singular integration. The collocation method consists in replacing the continuous functions ϕ and $\frac{\partial \phi}{\partial n}$ with their numerical approximations ϕ_h and γ_h , which represent the discretized potential and potential normal derivative respectively in the finite dimensional space, and collocating the BIE on a number of points equal to the number of unknowns. A natural location for these points is on the boundary Γ_h , selecting, for example, the support points of the basis functions. Another possible choice would be to collocate the equations close to, but not on the boundary, in such a way the integrals appearing in (1.5a) would not be singular at all. However, if the domain presents sharp corners or cusps the point selection may not be straight-forward. We collocate on the boundary Γ_h , using the support points of the Lagrangian Finite Element Spaces.

Collocating (1.5a) produces the linear system

$$(\alpha + N)\hat{\phi} - D\hat{\gamma} = 0, \quad (1.9)$$

where

- α is a diagonal matrix with the values $\alpha(x_i)$, where x_i represents the i -th collocation point;
- $N_{ij} = \sum_{k=1}^K \sum_q^{N_q} \frac{\partial G}{\partial n}(x_i - x_q) \psi_q^j J^k$, where \hat{K} represents the reference cell and J^k is the determinant of the first fundamental form for each panel k ;
- $D_{ij} = \sum_{k=1}^K \sum_q^{N_q} G(x_i - x_q) \psi_q^j J^k$;
- $\hat{\phi}, \hat{\gamma}$ represent the nodal value of potential and potential normal derivative.

The integrals on the reference cell are performed using different quadrature schemes. When the collocation point does not lie inside the cell we simply use bidimensional Gauss scheme, If the collocation point is inside the current cell we need a different strategy since the Kernels $G, \partial G/\partial n$ are singular, in particular we use bidimensional Telles or Lachat Watson quadrature formula, [114]. On one hand this choices allows for an accurate solution of the weakly singular Boundary Integral Equation, and on the other hand it makes the assemblage cycle more complicated. This is mainly due to the fact that we need to use two different discrete schemes checking if the collocation point lies inside the current cell.

Chapter 1. Laplace Boundary Integral Formulation

Imposition of the boundary conditions To impose the boundary condition for ϕ_h and γ_h , we define the additional vectors

$$\bar{\phi} = \begin{cases} \hat{\phi}_i = f_D(x_i) & \text{if } x_i \in \Gamma_D \\ 0 & \text{if } x_i \in \Gamma_N. \end{cases} \quad (1.10a)$$

$$\bar{\gamma} = \begin{cases} 0 & \text{if } x_i \in \Gamma_D \\ \hat{\gamma}_i = f_N(x_i) & \text{if } x_i \in \Gamma_N \end{cases} \quad (1.10b)$$

$$\tilde{\phi} = \begin{cases} 0 & \text{if } x_i \in \Gamma_D \\ \hat{\phi}_i & \text{if } x_i \in \Gamma_N. \end{cases} \quad (1.11a)$$

$$\tilde{\gamma} = \begin{cases} \hat{\gamma}_i & \text{if } x_i \in \Gamma_D \\ 0 & \text{if } x_i \in \Gamma_N. \end{cases} \quad (1.11b)$$

With the definitions (1.10) and (1.11) we can introduce the linear system

$$A\tilde{t} = b, \quad (1.12)$$

where b is recovered using (1.13), the linear operator A satisfies (1.14),

$$b = -(\alpha + N)\bar{\phi} + D\bar{\gamma}, \quad (1.13)$$

$$A\tilde{t} = (\alpha + N)\tilde{\phi} - D\tilde{\gamma}, \quad (1.14)$$

\tilde{t} simply groups the unknowns as in (1.15)

$$\tilde{t}_i = \begin{cases} \hat{\gamma}_i & \text{if } x_i \in \Gamma_D \\ \hat{\phi}_i & \text{if } x_i \in \Gamma_N. \end{cases} \quad (1.15)$$

We don't need to assemble the matrix A in (1.14) but we simply define its vector matrix multiplication on a generic vector, this is enough for an iterative Krylov solver. We need to take great care of the interfaces between different boundary condition in the case of mixed problem. In such case we treat the interface as a sharp edge and we impose different boundary condition on the corresponding double nodes.

Solution of the linear system The procedure above leads to a non symmetric (dense) linear system which is solved iteratively using a preconditioned GMRES Krylov solver. We use an incomplete Gauss factorization of a band matrix derived from the system matrix as a preconditioner. Algebraic Multi Grid preconditioners have proved to be very effective in the solving of BIEs with Neumann or Dirichlet boundary datum. We are currently studying an application of AMG preconditioners in the case of mixed boundary conditions. We have chosen to restart the solver every 30 iterations. By our choice of preconditioner, this is sufficient to find a solution with a reasonable number of iterations ($n < 100$).

Post Process In many applications we need to post process the potential ϕ and its normal derivative. In most cases we need to compute an error analysis to verify the proper convergence of the method, see Sections 1.3.1, 1.3.2 and 1.3.3. Another classical example arises in fluid-dynamics where the potential gradient $\nabla\phi$ represents the fluid velocity. We apply an L_2 projection of the surface gradient $\chi = \nabla_s\phi$ combined with the potential normal derivative $\frac{\partial\phi}{\partial n}$ to construct a proper *gradient recovery* post-processing step, i.e., we solve

$$\int_{\Gamma} v_h \chi \, ds = \int_{\Gamma} v_h \left(\nabla_s \phi + \frac{\partial \phi}{\partial n} \mathbf{n} \right) \, ds \quad \forall v_h \in V_h. \quad (1.16)$$

From (1.16) we get a $O(N)$ sparse system similar to the one we use to recover the normal vector on the nodes to be used in the imposition of the boundary conditions. In this scenario equation (1.16) only requires the inversion of a (boundary) sparse mass matrix while the right hand side is completely known after the BEM system resolution.

We present all the aforementioned features in an OpenSOURCE code whose structure is reported in Figure 1.1, we underline that the modular structure we have chosen allows for modification to the code. For instance it is quite straightforward to plug in an accelerator, if it satisfies the assumptions our presentation is built upon, instead of the real matrix assembling cycles.

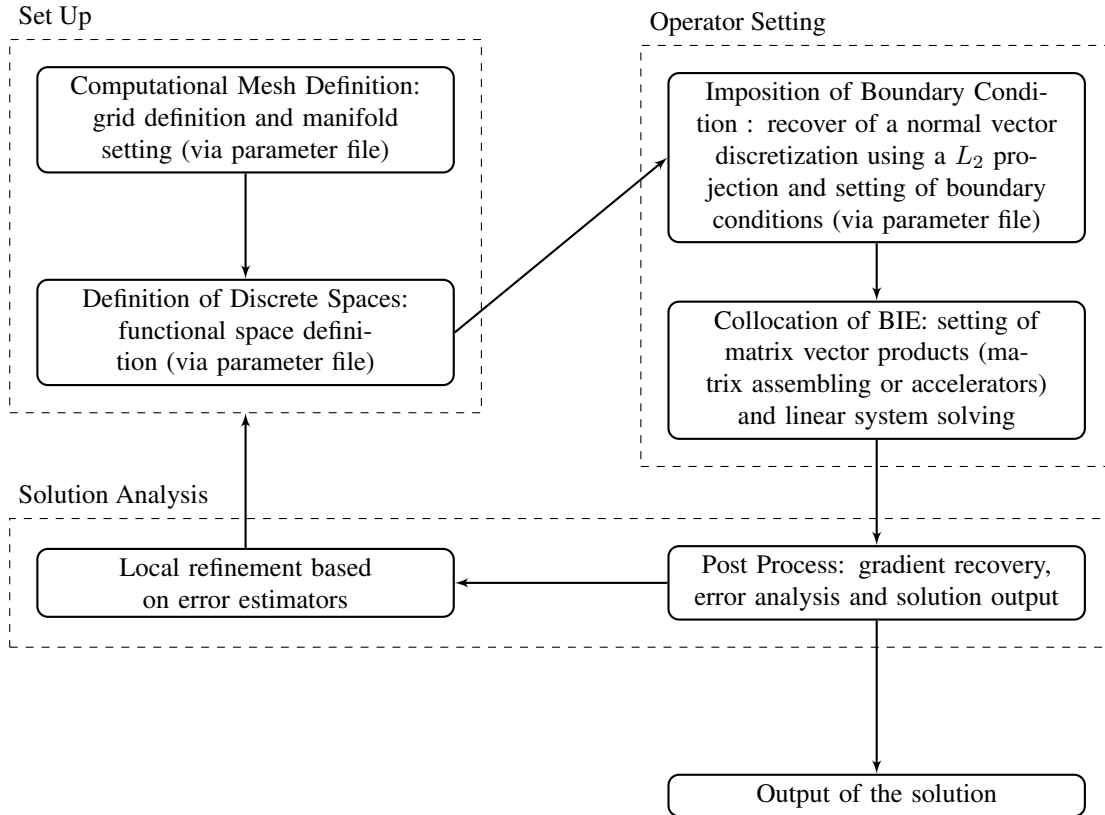


Figure 1.1: Flow diagram depicting the structure of the program for the solution of the Laplace BEM.

1.3 Results

1.3.1 Global refinement strategy

We analyse the convergence of the numerical scheme to a known solution for a mixed Dirichlet-Neumann problem on different domain configurations, by considering the following harmonic function as a non-trivial, known, analytical solution

$$\Phi = \frac{-1}{4\pi\sqrt{(x-1)^2 + (y-0.5)^2 + (z-0.5)^2}}, \quad (1.17)$$

and its corresponding gradient to solve a mixed Neumann Dirichlet Boundary value problem. As illustrated in Figure 1.2, the first geometry considered is that of a sphere of radius $R = 1$ and centered in $O = (0, 0, 0)$, in which both Dirichlet and Neumann faces are present. On such a prescribed geometry it is in principle possible to obtain the potential normal derivative to be imposed as Neumann boundary condition. In fact the normal is known at any point of the sphere, and so is the gradient of the potential Φ . Unfortunately in most complex cases the analytic normal vector to the specified geometry is not always known *a priori*. Before the assembling and resolution of the BEM system we introduce a pre processing step consisting in a discretized L_2 projection of the normal vector, resulting in a sparse system of $O(N)$. In cases where the gradient of Φ is analytically known, we obtain the value of Neumann boundary conditions as the product of such gradient by the computed normal. This is exploited to verify the accuracy of the normal derivative approximation.

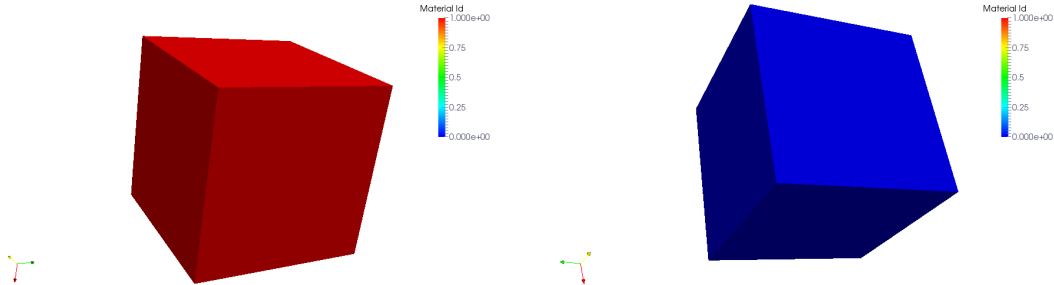


Figure 1.2: Initial Mesh configuration for the sphere test case with mixed boundary conditions. We show the initial grid, represented by a simple cube composed by 6 faces. Dirichlet boundary conditions are applied to the red faces, while Neumann boundary conditions are applied to the blue faces.

The convergence test is carried out starting from a very coarse representation of the domain which is refined until the number of unknowns of the resulting numerical problem leads to a dense matrix that cannot be stored using 32 bit indexing.

Figure 1.3 shows the first mesh refinement for which the results are meaningful and the final refined version. At each refinement level the new cells are created on the specified geometry. The possibility to introduce a very coarse discretization of the domain, which is then refined at will to represent at best the desired geometry, is a key feature of our implementation. Such feature is particularly important in this example, since the curved shape of the domain boundary is ideal to highlight the error due to the quadrature of both single and double layer kernels. We recall that for flat surfaces the

error in the quadrature of the latter kernel is negligible since

$$\frac{\partial G}{\partial n}(\mathbf{r}) = \frac{\mathbf{r} \cdot \mathbf{n}}{4\pi|\mathbf{r}|^3} = 0. \quad (1.18)$$

Figure 1.4 presents the resulting convergence analysis for both the potential ϕ (on the left) and its normal derivative (on the right). In the plots the green lines represent the L_2 norm of the error as a function of the number of unknowns. The blue lines indicate the corresponding value of the L_∞ norm while the red line is the reference first order convergence rate.

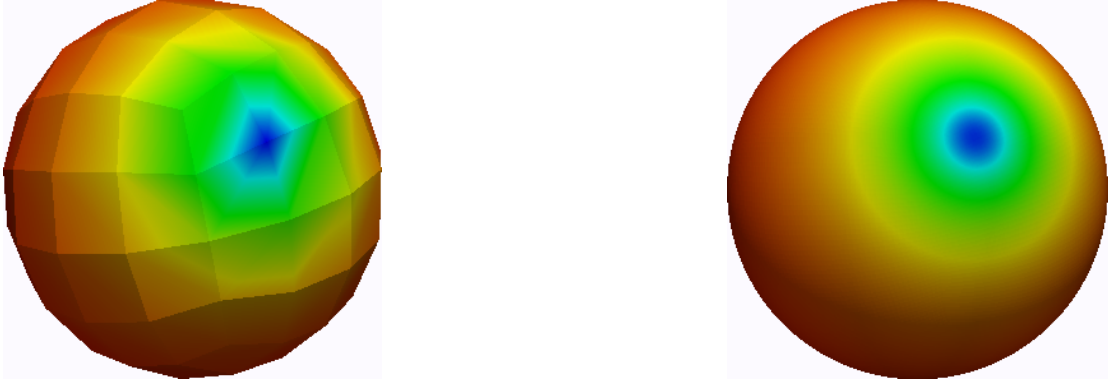


Figure 1.3: Initial and final refinement for the sphere test case. On the left we have considered 96 cells, corresponding to 150 unknowns, while on the right we have 98304 cells, corresponding to 99074 unknowns. The colours depict the magnitude of the exact solution ϕ .

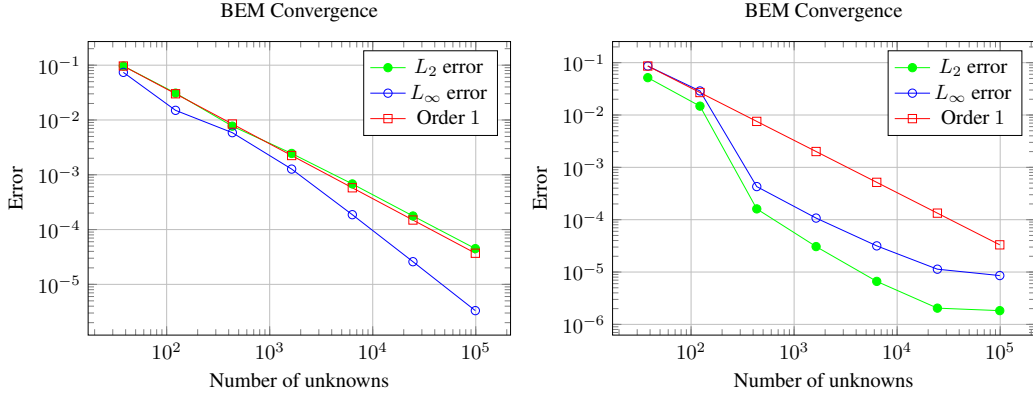


Figure 1.4: Convergence analysis for the error in a mix Dirichlet-Neumann problem using Q_1 boundary elements and the spherical mesh. On the left we plot the analysis for the variable ϕ on the right we depict the errors for $\partial\phi/\partial n$. The blue curve represents the L_∞ norm and the green one the L_2 norm. We have reported the first order convergence rate in red.

As clearly indicated in the left plot of Figure 1.4, the use of Q_1 elements results in a first order convergence for the variable ϕ , for both L_2 and L_∞ norm. As for $\frac{\partial\phi}{\partial n}$ convergence a more irregular behavior is observed. During the first refinement steps in fact, the error decreases more than linearly, while the convergence rate decreases significantly in correspondence with the final refinement step. The initial super-convergence

might be result of the spherical symmetry of the domain. A closer look at the error distribution (see Figure 1.5) suggests that the progressive refinement of an initially cubic mesh over a sphere results in the presence of few stretched cells located at the original mesh vertices. To make things worse such nodes are located at the interface between different boundary condition regions. This situation is neither altered nor improved throughout the refinements of the grid, as the angle of such stretched cells remain constant for each mesh. The stretched cells might result in a constant error which might become a dominant term for the finest discretization tested. To confirm such conjecture we will test in the close future different mesh blockings for the sphere test case.

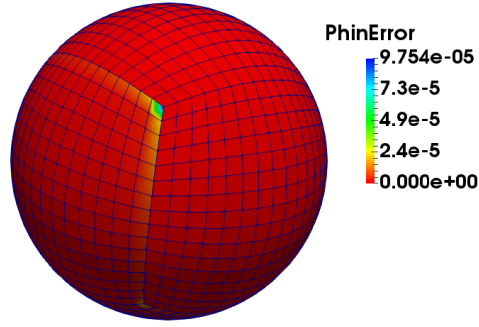


Figure 1.5: Error analysis for the potential normal derivative in the sphere test case. The maximum values are reported in blue. We see an error concentration where the cells are more stretched.

The second test case considered is the truncated pyramid illustrated in Figure 1.6. Also in this case we employ the analytical solution (1.17) to impose both Dirichlet and Neumann boundary conditions on different portions of the boundary. Given the previous considerations this test case is devised to minimize the error related to the singular kernel integrations so as to evaluate the error related to the presence of sharp corners characterized by different angles. In this case the first grids consists of 96 quadrilateral cells and it is refined up to 98304. The results of the convergence analysis are presented in Figure 1.7, both for the potential ϕ (on the left) and its normal derivative (on the right). In the plots the green lines indicate the L_2 norm of the error as a function of the number of unknowns. The blue lines represent the corresponding value of the L_∞ norm while the red line is the reference first order convergence rate.

Figure 1.7 shows that using standard Q_1 elements on a domain that does not present the symmetries of the sphere still results in first order convergence rate for the potential ϕ . In the case of the potential normal derivative we observe the occurrence of an error plateau for the L_∞ norm in corresponding to the last refinement level considered. A possible cause can be assessed through an inspection of the local $\frac{\partial \phi}{\partial n}$ error distribution presented in Figure 1.8. As can be seen the maximum error occurs in correspondence with the edge characterized by the sharpest angle. This leads us to infer that in presence of narrow corners an error might be the result of not applying a singular quadrature on one side of the corner when integrating a singularity located on the opposite site. Further investigation on the optimal application of singular quadrature formulas in presence of sharp edges is undergoing.

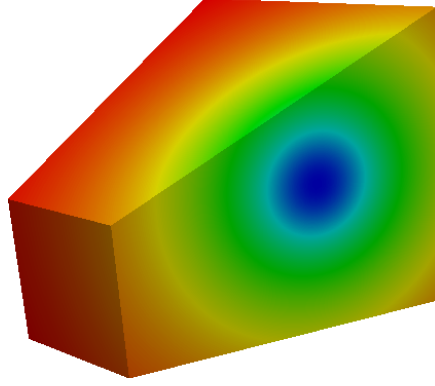


Figure 1.6: A sketch of the truncated pyramid geometry considered in the second test case. The colors represent the magnitude of the exact solution ϕ .

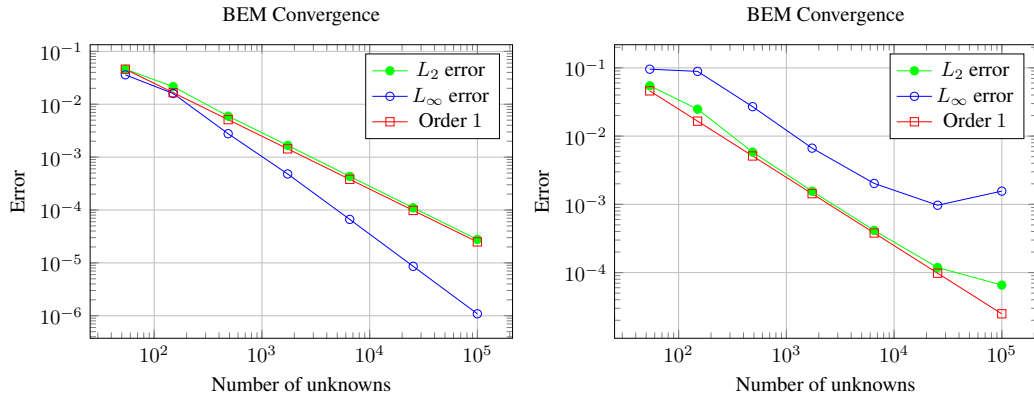


Figure 1.7: Convergence analysis for the error in a mix Dirichlet-Neumann problem using Q_1 boundary elements and the truncated pyramid mesh. On the left we plot the analysis for the variable ϕ on the right we depict the errors for $\partial\phi/\partial n$. The blue curve represents the L_∞ norm and the green one the L_2 norm. We have reported the first order convergence rate in red.

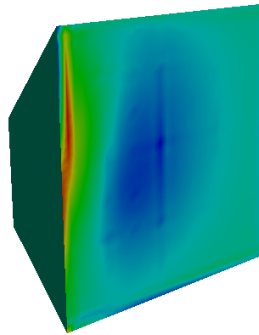


Figure 1.8: Error analysis for the potential normal derivative in the pyramid test case. The maximum values are reported in blue. We see an error concentration due to the sharp edge together with some residual integration errors.

1.3.2 Local refinement strategy

The developed library allows for the refinement of a simple and coarse mesh over a desired geometry. In the previous examples such feature was used in the framework of a global refinement strategy. We present here its effectiveness in conjunction with local adaptive refinement strategies. Such strategies consist in refining the original mesh by distributing a major number of cells in the regions where the solution is less accurate. In this Section we will present the results of the application of adaptive refinement strategy based on Kelly Error Estimator to the sphere test case discussed in Section 1.3.1. The adaptive refinement procedure is started from the uniform coarse grid shown on the left portion of Figure 1.9. At each refinement cycle we use the computed solution for the potential ϕ to evaluate the Kelly error estimator for each cell of the domain. Based on such estimator we proceed to refine the 30% of the total cells having the highest values of the error estimator. Since each refined cell is split into four children this procedure results in roughly doubling the total amount of mesh elements at each refinement step. The right plot of Figure 1.9 presents the adapted mesh obtained after 8 local refinement cycles. The plot confirms that by a qualitative standpoint the refinement algorithm is able to concentrate the computational grid nodes in the regions where the solution gradient is highest. The results of the convergence analysis are presented in Figure 1.10 for the potential ϕ . In the plots the green lines indicate the L_2 norm of the error as a function of the number of unknowns. The blue lines represent the corresponding value of the L_∞ norm. The continuous lines denote the error obtained with adaptive local refinement strategy, while the dashed lines represent the corresponding global refinement strategy errors. The latter errors have been already presented in Figure 1.7 and are here reported as a comparison reference for the adaptive refinement.

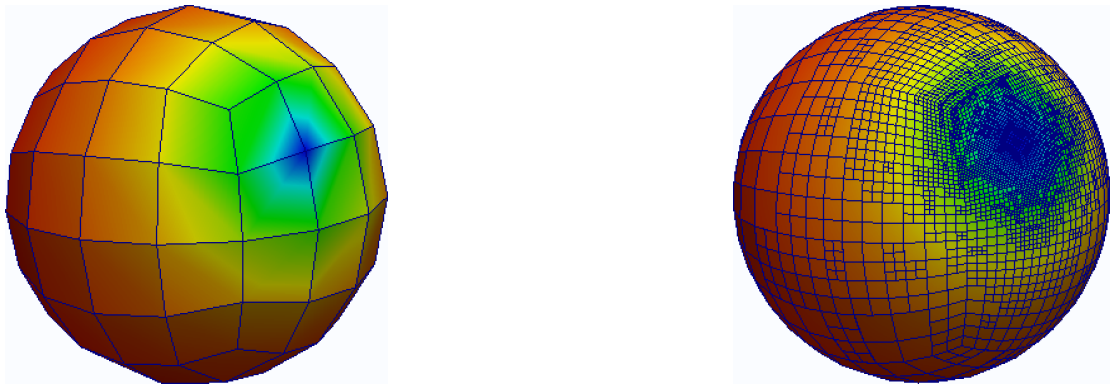


Figure 1.9: Initial and final refinement for the sphere test case using local refinement strategies. On the left we have considered 96 cells, corresponding to 150 unknowns, while on the right we have 19515 cells, corresponding to 20410 unknowns. Also in this case the colors represent the magnitude of the exact solution for the variable ϕ

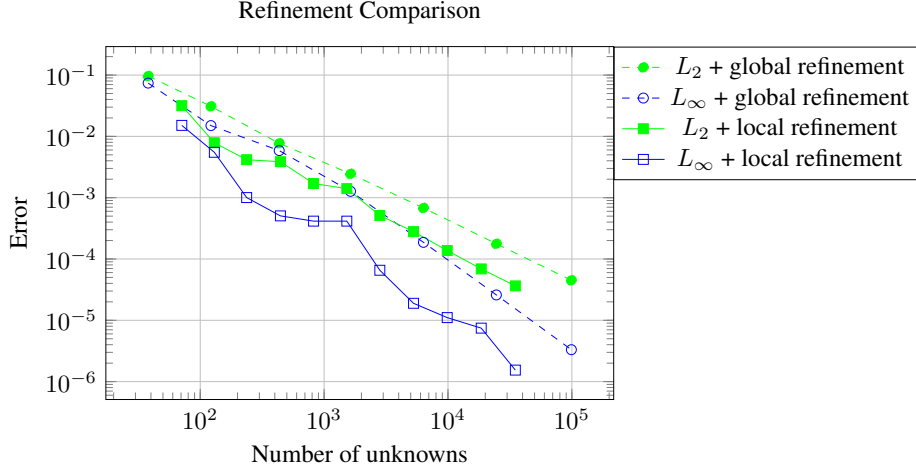


Figure 1.10: Convergence analysis for the error on ϕ on the sphere in a mixed Dirichlet-Neumann problem using Q_1 boundary elements. In blue we see the L_∞ norm and in green the L_2 norm.

The results in the convergence analysis confirm that the adaptive refinement strategy converges to the correct solution with the same order. Yet given the same number of mesh elements the errors obtained are significantly lower with respect to the global refinement case. In the future we will further investigate on the performance of different error estimators which could result in a further reduction of the computational costs. Nonetheless, a simple indicator such as Kelly error estimator is already giving a significant contribution in this regard.

1.3.3 High order basis functions

To test the performance of different choices for the Lagrangian basis function used in the iso-parametric discretization scheme implemented in the present work, we carry out a further convergence analysis on the pyramid test case. Such geometry in fact might in principle be less indicated for high order and finite elements. In fact sharp corners might reduce the possibility to exploit such elements which are effective with highly regular solutions. Yet the implementation of the double nodes treatment of sharp edges, allows for a full exploitation of the high order discretization advantages even on irregular geometries. Thus, the test consists in a convergence analysis based on global refinement cycles for bi-linear (Q_1) bi-quadratic (Q_2) and bi-cubic (Q_3) quadrilateral finite elements. Figure 1.11 presents the results of such analysis for both ϕ (on the left) and its normal derivative (on the right). The circled blue lines represents the error using bilinear Q_1 elements, the squared green ones the error using biquadratic Q_2 elements, while the red plots show the error using bicubic elements. The errors obtained with Q_1 elements have been already presented in Figure 1.7 and are here reported as a comparison reference for the high order global refinement.

The potential function convergence plot suggests that quadratic and cubic elements converge to the exact solution at higher convergence rate with respect to the linear discretization. Indeed second order elements lead to convergence rate of 1.5, whereas cubic elements converge with order 2. The final refinement step considered in this analysis displays a slight reduction in the quadratic and cubic elements convergence rate. This is probably caused by an even more evident accuracy loss which can be

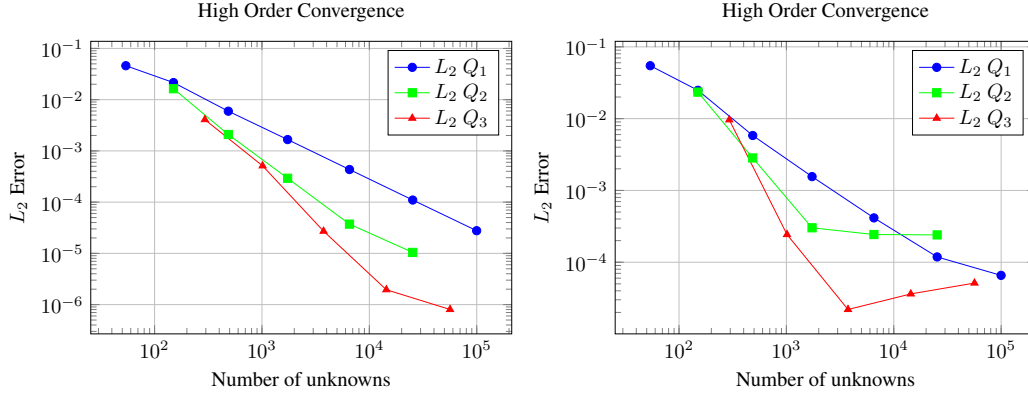


Figure 1.11: Convergence analysis for the error in a mix Dirichlet-Neumann problem using boundary elements and the truncated pyramid mesh. On the left we analyze the convergence of the solution ϕ and on the right its normal derivative. In blue we plot the result using Q_1 elements, in green we report the Q_2 analysis and in red we draw the results using Q_3 finite elements.

observed in the $\frac{\partial \phi}{\partial n}$ plot.

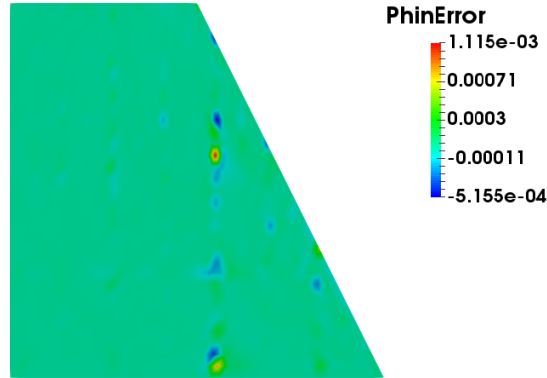


Figure 1.12: Error analysis for the potential normal derivative in the pyramid test case using bicubic Q_3 elements. The maximum values are reported in blue. We see an error concentration due to the sharp edge together with some residual integration errors.

The cubic error distribution presented in Figure 1.12 displays a rather chequered pattern suggesting that the inaccuracy might be related to the integration of singular kernels. This might be explained with the fact that quadratic and cubic shape functions assume negative values on each element resulting in possible non negligible cancellation errors in sums involving very high $O(\frac{1}{r})$ terms. Up to this moment we have used Lachat–Watson or Telles quadrature scheme, see [114], up to order 50. We are currently studying improvements to the quadrature schemes which might increase the accuracy of weakly singular integrals.

Despite such problems occurring at the last stages of refinement, the use of high order discretizations represent a very interesting alternative. The convergence plots show that quadratic and cubic basis functions reach errors of the order 10^{-3} or 10^{-4} with significantly less degrees of freedom with respect to the linear discretization. This

can represent a clear advantage in several engineering applications in which errors of this size are considered acceptable.

A further quantitative confirmation to these considerations can be obtained by a different test case employing the same truncated pyramid geometry used in the previous one. Here we apply local adaptive refinement cycles until L_∞ norm of the error on the potential function ϕ reaches a value lower than 5×10^{-5} . Table 1.1 presents the results of such analysis for Q_1, Q_2, Q_3 elements.

Table 1.1. All the cases were started from a uniform coarse grid which has been adaptively refined.

lement	$L_\infty \phi$ Abs Error	Cells	Unknowns	Time (sec)
Q_1	4.52071×10^{-5}	2559	2877	40.98
Q_2	2.6995×10^{-5}	327	1576	12.36
Q_3	2.315×10^{-5}	165	1763	42.12

Table 1.1: Comparison among the performance of continuous Finite Elements of different degree coupled with local refinement strategy. The Neumann–Dirichlet mixed problem is recursively solved on the sphere and the computational grid is adaptively refined until the prescribed 5×10^{-5} error threshold is reached. The table reports both the number of unknowns of the final grid and the total computational time needed to reach the prescribed error.

The data show that biquadratic high order elements allow for a remarkable reduction of the degrees of freedom required to obtain the prescribed accuracy, this results in a computational time reduction of roughly 2 and 4 for quadratic and cubic elements, with respect to linear ones. However we see that the usage of bicubic elements cause an increase in the computational time, this is due to the weakly singular kernel integrations. A very accurate integration scheme is necessary to properly integrate such singularities when convolved with highly oscillating functions. From Figure 1.11 we see that high order elements are able to provide extremely accurate solutions but we need to take into account the computational cost to properly implement the weakly singular integrals.

1.3.4 Complex geometries integration

In Section 1.3.2 we have shown that the implementation we propose is capable of refining over a prescribed geometry, and in Section 1.3.1 we have shown that we are able to treat sharp edges in the domain. At this point we want to use the Laplace solver on an arbitrarily complex geometry of industrial interest. In the last decades the developing of Computer Aided Design technology has deeply influenced both FEM and BEM communities. Especially in Boundary Element Methods the possibility of integrating the exact geometry opens up new possibilities, see [79]. We provide our implementation with the possibility of using different kind of manifold. In particular the user can specify an arbitrary number of surfaces and curves imported through `iges` files. This is essential in order to treat geometries of industrial interest. In Figure 1.13 we present an example using a CAD file describing a NACA0012 airfoil. While this is still a simple situation, it presents all the challenges of working with a complex geometry of industrial interest. In fact, the domain presents sharp edges and a cusp at the trailing edge. We are able to deal with such complexity integrating the aforementioned double nodes treatment, local refinement strategies and CAD geometry descriptors. The only requirement for the simulation set up is an isotropic mesh with nodes located on the

imported geometry. Once this is provided all the —local or global— refinements required will be carried out on the desired domain. In the case at end, the NACA airfoil test consists in a convergence analysis based on global refinement cycles for bi-linear (Q_1) quadrilateral finite elements. Also in this case the analytical solution used for such convergence analysis is the potential Φ represented in (1.17). Figure 1.13 presents the computational mesh we are using. The left image represents the starting isotropic mesh. The right plots depicts the computational mesh after two fully automated global refinement cycles. Both plots are colored according to the local values of the computed potential ϕ .



Figure 1.13: On the left we plot the starting NACA0012 computational mesh (637 dofs). On the right the computational mesh after two global refinements (8395 dofs). The contour depicts the solution for the potential ϕ as in (1.17).

Figure 1.14 depicts the results of the convergence analysis. The plot on the left depicts the study potential ϕ both in L_2 (blue empty circles) and L_∞ (green full circles) norms, while the Figure on the right represents the analysis for the potential derivative $\frac{\partial \phi}{\partial n}$ in L_2 (blue empty circles) and L_∞ (green full circles) norms. We have reported the first order convergence rate in red.

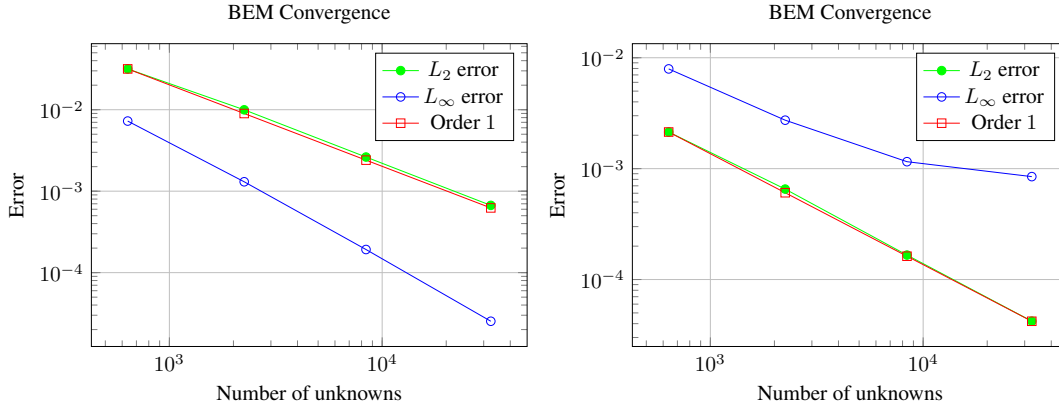


Figure 1.14: Convergence analysis for the error in a mix Dirichlet-Neumann problem using Q_1 boundary elements and the truncated pyramid mesh. On the left we plot the analysis for the variable ϕ on the right we depict the errors for $\partial \phi / \partial n$. The blue curve with empty circles represents the L_∞ norm and the green one with full circles the L_2 norm. We have reported the first order convergence rate in red.

The left plot in Figure 1.14 suggests that the use of bilinear quadrilateral elements results in the same first order convergence reported in Section 1.3.1 and 1.3.2 for the

potential ϕ . As for the potential normal derivative the convergence in L_∞ norm progressively slows down in correspondence with the last refinement cycles. A first cause for this behavior could be related to the presence of distorted cells on the lateral planar faces in proximity with the cusp. In such region, regardless of the mesh refinement or quality the cell on the cusp will present a $\sim 15^\circ$ angle which likely leads to an error independent of refinement. A further source of error is possibly related to the use of singular quadrature only on the cells where each collocation node is located. In the last cycles, in correspondence with the cusp and with progressively finer grids the error of not accounting for the singularity on the opposite sides of the cusp with suitable quadratures might become relevant.

This test suggests that with bilinear elements we retain linear convergence to a non trivial solution on a complex geometry if the primary variable ϕ is considered. As for $\frac{\partial\phi}{\partial n}$, the behavior is sublinear in correspondence with the last steps of refinement. Nonetheless, even in this geometrically challenging problem the minimum L_∞ error obtained is below 10^{-3} which is perfectly acceptable in many industrial applications.

We are currently developing a second kind of adaptive local refinement strategy which is exclusively based on the local curvature of the imported CAD shape. In many cases in fact it is important that the computational mesh employed in the simulation is able to represent in the best possible way the actual geometry. Comparisons between such strategy and the one presented in Section 1.3.2 are undergoing.

1.3.5 Post process and gradient recovery

In many cases the variable of interest is neither the potential ϕ nor its normal derivative, but a function depending on both. Thus, it is essential to correctly Post Process the solution of the BEM solver. For example the potential gradient $\nabla\phi$ often represents a meaningful quantity, in fluid dynamics it describes the velocity of the fluid moving around the domain Γ . As described in Section 1.2 a L_2 projection is implemented to properly recover the overall gradient combining the surface gradient $\nabla_s\phi$ with its normal component $\frac{\partial\phi}{\partial n}$. Figure 1.15 represents the potential gradient $\nabla\phi$ obtained solving a full Neumann problem considering external aerodynamics around the NACA0012 airfoil depicted in Section 1.3.4. We impose an asymptotic uniform flow directed along the x axis. We see that the glyphs representing the fluid velocity are tangent to the wing profile. This is expected for the non penetrating Neumann boundary condition we impose.

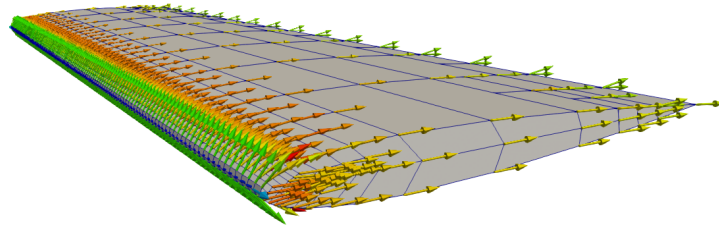


Figure 1.15: *Post Process of a BEM solution regarding external aerodynamics around a NACA0012 airfoil. The glyphs represent the complete potential gradient $\nabla\phi$. We impose an asymptotic uniform flow directed as the x axis and we recover a velocity tangential to the wing itself.*

CHAPTER 2

Linearized ship-wave interaction problems using Boundary Element Method and FEM-SUPG stabilization

We focus on a physical application of the Boundary Element Method for the Laplace equation developed in Chapter 1. We are interested in studying the generation of gravitational waves around a body advancing at constant speed in calm water. This type of waves presents a typical pattern, called the Kelvin-wake pattern, which is characterized by a strongly V-shaped system. In section 2.2.1 we will focus on the waves generated by a fully submerged spheroid (Figure 2.1), while in section 2.2.2 we will study the waves generated by a surface piercing body (Figure 2.2). Both problems are assessed benchmarks in naval engineering (see, e.g., [29,85,98]). In all simulations we consider a flow domain composed by the portion of water surrounding the body, as depicted in Figures 2.1 or 2.2. The flow domain is bounded by the free surface, by the body and by the bottom surfaces. The part of the boundary Γ_∞ represents the truncation surfaces of the numerical domain, which is considered to be far enough from the body. If the body pierces the free surface, as in Fig. 2.2, the domain includes only the part of the body beneath the water surface, excluding its dry part.

We consider the flow of an incompressible inviscid fluid past a body at rest, or, equivalently, that of a body moving at constant speed in a fluid at rest, in a frame of reference attached to the body. We choose to use the Euler equations for incompressible fluids. In this case the mass conservation can be rewritten as the classic scalar Laplace equation. We complete the problem by selecting the boundary conditions to be prescribed on the remaining boundary regions. On the body boundary Γ_{body} we impose a —non-homogeneous Neumann— non penetration condition. Homogeneous Neumann conditions are instead imposed on the lateral truncation surfaces Γ_{tank} of the

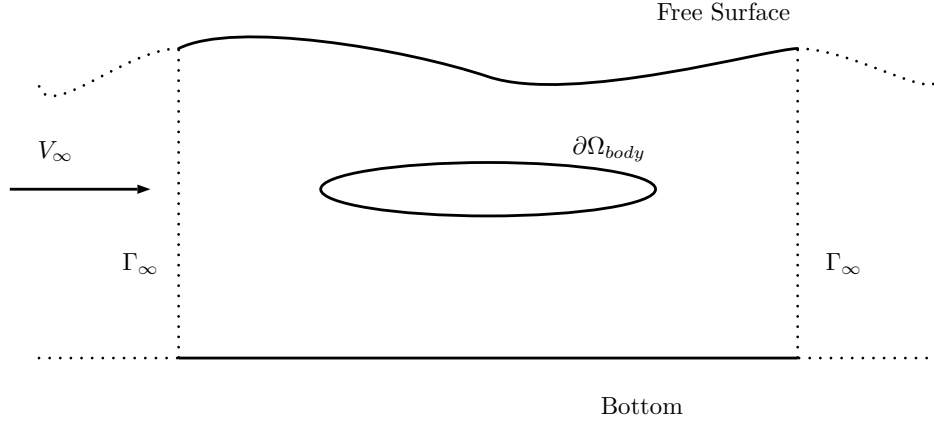


Figure 2.1: Vertical section of the domain for the simulation of the flow past a body beneath the water free surface.

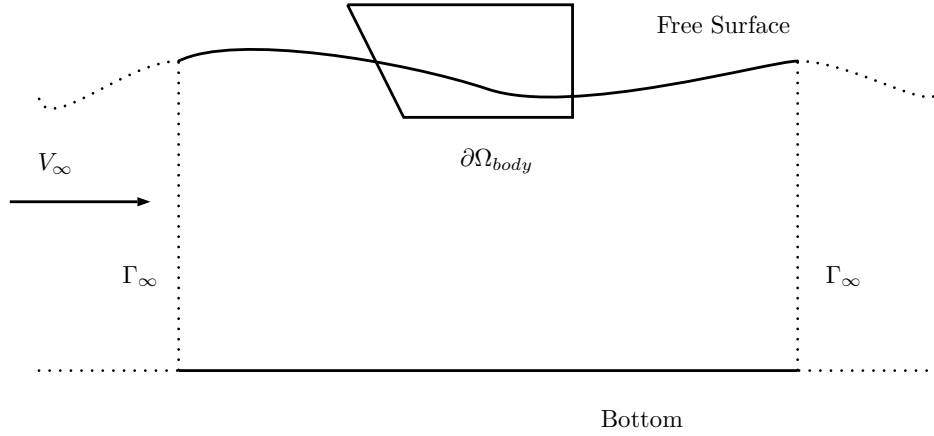


Figure 2.2: Vertical section of the domain for the simulation of the flow past a boat located across the water free surface

numerical tank. The same condition is imposed on the outflow boundary Γ_{out} , while a homogeneous Dirichlet boundary condition is imposed on the inflow boundary Γ_{in} . The latter condition is selected in order to avoid a pure Neumann boundary problem, the solution of which would be defined only up to a constant. The complete boundary value problem reads:

$$-\Delta\phi = 0 \quad \text{in } \Omega \quad (2.1a)$$

$$\frac{\partial\phi}{\partial n} = -\mathbf{v}_\infty \cdot \mathbf{n} \quad \text{on } \Gamma_{body} \quad (2.1b)$$

$$\frac{\partial\phi}{\partial n} = -\frac{U_\infty^2}{g} \frac{\partial^2\phi}{\partial x^2} \quad \text{on } \Gamma_{fs} \quad (2.1c)$$

$$\frac{\partial\phi}{\partial n} = 0 \quad \text{on } \Gamma_{out} \cup \Gamma_{tank} \quad (2.1d)$$

$$\phi = 0 \quad \text{on } \Gamma_{in}. \quad (2.1e)$$

For further details on the derivation of the free surface boundary condition (2.1c) see Section 2.1. It must be pointed out that the homogeneous Neumann conditions prescribed on the truncation surfaces might result in an undesired reflection of water waves back in the flow domain (see [107]). To limit this problem, in this work we employed a computational domain of considerable dimensions. The lateral truncation boundaries Γ_{tank} are located in fact at a distance $15 \times L$ from the origin, while the Γ_{in} and Γ_{out} surfaces are at distances $15 \times L$ and $15 \times L$ from the origin respectively, where L is the length of the body.

2.1 Free surface boundary condition

We now briefly discuss the boundary condition to be imposed on the free surface following what is presented in [85]. We consider a main flow velocity \mathbf{v}_∞ defined as $\mathbf{v}_\infty = U_\infty \mathbf{e}_x$ where \mathbf{e}_x identifies the x axis of our domain. We assume, as in [78, 85], that it is possible to represent the free surface elevation as a Cartesian function (thus excluding breaking waves), in which the z coordinate is a single valued function of the horizontal coordinates x, y :

$$\varsigma(x, y, z) = z - \eta(x, y) = 0. \quad (2.2)$$

By a kinematic stand point, one requires that fluid particles on the free surface will remain on the free surface, and fulfill equation (2.2). By a dynamical point of view instead, the water pressure on the free surface must always be equal to the air atmospheric pressure P_a , which is assumed constant and uniform. These two conditions can be expressed as

$$\frac{D}{Dt}(\varsigma) = \frac{D}{Dt}(z - \eta) = 0 \quad (2.3)$$

and

$$\frac{1}{2} \nabla \Phi \cdot \nabla \Phi + g\eta = C. \quad (2.4)$$

The first expression is the so-called kinematic boundary condition and it states that the material derivative of the quantity $z - \eta$ vanishes on the free surface boundary. The second, dynamic, condition is obtained from Bernoulli equation evaluated on the free surface. In the dynamic Bernoulli equation, C is an arbitrary constant in case of steady state solutions, and can be fixed by imposing the atmospheric pressure and evaluating

(2.4) at infinity, where the flow is assumed uniform, and both the perturbation potential ϕ , and the free surface elevation η vanish. We obtain

$$C = \lim_{|\mathbf{x}| \rightarrow \infty} \left(+\frac{1}{2} \nabla \Phi(\mathbf{x}) \cdot \nabla \Phi(\mathbf{x}) + g\eta(x, y) \right) = \frac{1}{2} U_\infty^2. \quad (2.5)$$

2.1.1 Linearized boundary condition

A perturbation technique allows us to obtain a single, linearized, free surface boundary condition from equations (2.3) and (2.4). In equation (2.4) there are two unknown quantities, *i.e.*, Φ and η . We consider small perturbations of Φ and η , with respect to the undisturbed asymptotic flow given by $\Phi_0 = \mathbf{v}_\infty \cdot \mathbf{x}$ and $\eta_0 = 0$. The asymptotic expansions read

$$\Phi = \Phi_0 + \epsilon \Phi_1 + O(\epsilon^2) \quad (2.6)$$

$$\eta = \eta_0 + \epsilon \eta_1 + O(\epsilon^2). \quad (2.7)$$

Plugging these expansions into equation (2.3) and equating terms of order ϵ , we obtain

$$\frac{\partial \Phi_1}{\partial z} - \nabla \Phi_0 \cdot \nabla \eta_1 - \nabla \Phi_1 \cdot \nabla \eta_0 = 0, \quad (2.8)$$

$$\nabla \Phi_0 \cdot \nabla \Phi_1 + g\eta_1 = 0. \quad (2.9)$$

We can write $\nabla \Phi_0 = \mathbf{v}_\infty$, $\phi = \epsilon \Phi_1$ and $\eta = \epsilon \eta_1$, leading to

$$\frac{\partial \phi}{\partial z} - \mathbf{v}_\infty \cdot \nabla \eta = 0, \quad (2.10)$$

$$\mathbf{v}_\infty \cdot \nabla \phi + g\eta = 0. \quad (2.11)$$

From equation (2.11) we see clearly that the imposition of the linearized boundary condition requires the resolution of a transport problem. This kind of problem often needs some kind of stabilization. We will discuss this topic in section 2.1.4. If we consider the steady state solution and if we assume that $\mathbf{v}_\infty = U_\infty \mathbf{e}_x$, the two linearized equations can be combined together as

$$U_\infty^2 \frac{\partial^2 \phi}{\partial x^2} + g \frac{\partial \phi}{\partial z} = 0 \quad \text{at} \quad z = 0. \quad (2.12)$$

Following [85] we apply equation (2.12) to the perturbation potential ϕ . This is a reasonable approximation if we assume that only small waves are generated, *i.e.*, if the wave amplitude A is much smaller than its wavelength λ ($A \ll \lambda$). If we exploit the fact that the undisturbed free surface is flat, then taking the partial derivative along the z direction is equivalent to taking the normal derivative, and the final boundary condition we obtain is

$$U_\infty^2 \frac{\partial^2 \phi}{\partial x^2} + g \frac{\partial \phi}{\partial n} = 0, \quad (2.13)$$

which is what we impose on the undisturbed free surface. The free surface elevation can be computed by postprocessing the flow potential through equation (2.11):

$$\eta = -\frac{U_\infty}{g} \frac{\partial \phi}{\partial x}. \quad (2.14)$$

2.1.2 Treatment of the linearized free surface condition

The presence of second order derivatives in the right hand side of the free surface boundary condition requires a careful treatment. Consider, for example, the case of linear finite dimensional spaces $V_h = Q_h$: in this case, the second order derivatives on each panel would be identically zero.

A solution consists in introducing an additional variable β (which we assume to be in the same space V of the potential ϕ), representing $\frac{\partial \phi}{\partial x}$, and to recover both the variable itself and its derivative using a weak formulation. Namely:

$$\int_{\Gamma} \beta v \, d\Gamma = \int_{\Gamma} \frac{\partial}{\partial x} \phi v \, d\Gamma \quad \forall v \in V \quad (2.15)$$

$$\int_{\Gamma} \gamma u \, d\Gamma = -\frac{U_{\infty}^2}{g} \int_{\Gamma} \frac{\partial}{\partial x} \beta u \, d\Gamma \quad \forall u \in Q. \quad (2.16)$$

The function β appearing in equations (2.15) and (2.16) is approximated as

$$\beta_h(x) = \sum_{i=1}^{N_V} \psi_i(x) \beta_i. \quad (2.17)$$

Using the finite dimensional spaces, equation (2.15) can be recast in the following form

$$M^V \hat{\beta} - B^V \hat{\phi} = 0, \quad (2.18)$$

where

$$M_{ij}^V = \int_{\Gamma} \psi_j(x) \psi_i(x) \, d\Gamma, \quad i, j = \{1, \dots, N_V\} \quad (2.19a)$$

$$B_{ij}^V = \int_{\Gamma} \frac{\partial \psi_j(x)}{\partial x} \psi_i(x) \, d\Gamma, \quad i, j = \{1, \dots, N_V\}. \quad (2.19b)$$

In the present work, we used the same approximation space for β and for the potential ϕ , even though the spaces could in principle be different. The additional variable β adds to the global algebraic system a set of N_V sparse rows obtained by the discretization of equation (2.15). This system represents an additional block in the complete BEM system. The three unknown vectors of the system are now $\hat{\phi}, \hat{\gamma}, \hat{\beta}$. Along the same lines, the discretized form of equation (2.16) assumes the form

$$M^Q \hat{\gamma} + \frac{U_{\infty}^2}{g} B^{QV} \hat{\beta} = 0, \quad (2.20)$$

where M^Q is the classic mass matrix, while the elements of B^{QV} are given by

$$B_{aj}^{QV} = \int_{\Gamma} \frac{\partial \psi_j(x)}{\partial x} \psi_a(x) \, d\Gamma, \quad a = \{1, \dots, N_V\}, \quad j = \{1, \dots, N_V\}. \quad (2.21)$$

The lines of this system are substituted to the lines of the BEM subsystem corresponding to the collocation points of the normal derivative approximation that are located on the free surface. Another derivation in weak form is applied to $\hat{\beta}$, which is then

projected using an L^2 projection in the space Q_h to obtain a weak form of the second derivative ($\partial^2 \phi / \partial x^2$) appearing in the right hand side of the linearized free surface boundary condition (2.16), in the same space of the normal derivative of the potential (Q_h). The additional variable $\hat{\beta}$ can be easily eliminated by inverting the mass matrix M^V and setting

$$C^Q := \frac{U_\infty^2}{g} B^{QV} (M^V)^{-1} B^V, \quad (2.22)$$

consequently (2.15) and (2.16) become

$$M^Q \hat{\gamma} + C^Q \hat{\phi} = 0. \quad (2.23)$$

2.1.3 Full resolution technique

To properly impose (2.13) we decide to apply a slightly different resolution strategy with respect to Section 1.2. A discrete form of the BIE (1.5a) is obtained, as in Section 1.2 by replacing the continuous solutions ϕ and $\frac{\partial \phi}{\partial n}$ by their finite dimensional approximations ϕ_h and γ_h , and imposing the original boundary integral equation at a sufficient number of *collocation points*. Such collocation points are placed in correspondence with the N_V support points of both the spaces V_h and Q_h . Thus we obtain a system of $N_V + N_V$ algebraic equations which reads respectively,

$$\begin{bmatrix} N^V & D^V \\ N^Q & D^Q \end{bmatrix} \begin{Bmatrix} \hat{\phi} \\ \hat{\gamma} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}, \quad (2.24)$$

where

$$\hat{\phi} = \{\phi_1, \dots, \phi_{N_V}\}, \hat{\gamma} = \{\gamma_1, \dots, \gamma_{N_V}\}.$$

are the vectors containing the unknown values of the approximated functions ϕ_i and γ_i at each collocation point. In the block system (2.24), the matrix rows in the top blocks are the ones obtained collocating the BIE on the N_V support points corresponding to the potential degrees of freedom, while the matrix rows in the bottom blocks are obtained using the N_V support points of the normal derivative as collocation points.

Given a collocation point x_i of the potential ϕ , the block matrix entries read, following Section 1.2,

$$N_{ij}^V = \alpha(x_i) + \sum_{k=1}^K \sum_q^{N_q} \frac{\partial G}{\partial n}(x_i - x_q) \psi_q^j J^k \quad (2.25a)$$

$$D_{ib}^V = \sum_{k=1}^K \sum_q^{N_q} G(x_i - x_q) \psi_q^j J^k \quad (2.25b)$$

while if x_i is a γ collocation point, we have

$$N_{aj}^Q = \alpha(x_i) + \sum_{k=1}^K \sum_q^{N_q} \frac{\partial G}{\partial n}(x_i - x_q) \psi_q^j J^k, \quad (2.25c)$$

$$D_{ab}^Q = \sum_{k=1}^K \sum_q^{N_q} G(x_i - x_q) \psi_q^j J^k, \quad (2.25d)$$

2.1. Free surface boundary condition

where K is the total number of elements Γ_h , J^K is the Jacobian of the mapping in the K -th element and the indices i, j run from one to N_V .

Notice that usually, on any given region of Γ , either ϕ is known and $\gamma = \frac{\partial \phi}{\partial n}$ is unknown, or the opposite. The free surface condition (equation 2.13) is an exception to this rule, and can be seen as special case of Robin boundary condition. On the free surface both ϕ and $\frac{\partial \phi}{\partial n}$ are unknown, but we impose a linear relationship between $\frac{\partial \phi}{\partial n}$ and a differential operator on ϕ .

To fix the ideas, consider the boundary Γ as the union of three disjoint sets, consisting of the Neumann boundary portion $\Gamma_N = \Gamma_{body} \cup \Gamma_{out} \cup \Gamma_{tank}$, the Dirichlet boundary portion $\Gamma_D = \Gamma_{in}$ and the Robin boundary portion $\Gamma_R = \Gamma_{fs}$. Both ϕ and γ can then be separated into their Neumann, Dirichlet and Robin parts respectively, according to the location of the support points.

System (2.24) can be solved explicitly by imposing the boundary conditions instead of the boundary integral equations in the appropriate lines of system (2.24):

$$\begin{bmatrix} N^V_{NN} & N^V_{ND} & N^V_{NR} & D^V_{NN} & D^V_{ND} & D^V_{NR} \\ 0 & I_{DD} & 0 & 0 & 0 & 0 \\ N^V_{RN} & N^V_{RD} & N^V_{RR} & D^V_{RN} & D^V_{RD} & D^V_{RR} \\ 0 & 0 & 0 & I_{NN} & 0 & 0 \\ N^Q_{DN} & N^Q_{DD} & N^Q_{DR} & D^Q_{DN} & D^Q_{DD} & D^Q_{DR} \\ 0 & 0 & C^Q_{RR} & 0 & 0 & M^Q_{RR} \end{bmatrix} \begin{Bmatrix} \hat{\phi}_N \\ \hat{\phi}_D \\ \hat{\phi}_R \\ \hat{\gamma}_N \\ \hat{\gamma}_D \\ \hat{\gamma}_R \end{Bmatrix} = \begin{Bmatrix} 0 \\ \bar{\phi}_D \\ 0 \\ \bar{\gamma}_N \\ 0 \\ 0 \end{Bmatrix}, \quad (2.26)$$

where M^Q_{RR} and C^Q_{RR} are the matrices C^Q and M^Q introduced in (2.23). Notice that the boundary integral equations in system (2.26) are evaluated only once in each region, and that, in principle, one could solve Robin's boundary condition by either imposing $\hat{\phi}_R$ and extracting $\hat{\gamma}_R$ from the linear combination, or by imposing $\hat{\gamma}_R$ and extracting $\hat{\phi}_R$. This alternative approach would result in line 6 of system (2.26) to be swapped with line 3, evaluating the boundary integral equations on support points of Q_h instead of V_h , i.e., replacing N^V and D^V by N^Q and D^Q respectively.

2.1.4 Streamwise Upwind Petrov Galerkin stabilization

The numerical strategy presented in the previous sections generates an accurate approximation of the second order derivative appearing in the linearized free surface boundary condition. Yet, this is not sufficient to obtain a physically meaningful approximation of problem (2.1), due to two issues of different nature: i) the boundary condition (2.13) is symmetrical in the x direction and ii) the problem is transport dominated.

If a physical wave propagating downstream with respect to the stream velocity \mathbf{v}_∞ fulfills the linearized condition, also an unphysical wave propagating upstream will satisfy such condition. Several different methodologies have been developed over the years, to selectively suppress the nonphysical solution without affecting the physical waves. In this work, we impose a homogeneous Dirichlet condition on the inflow surface of the domain, and a homogeneous Neumann condition on the outflow, as suggested in [107]. This setup satisfies a radiation condition and breaks the symmetry of the solutions in the x direction, privileging only the physical solution.

While in principle this should be sufficient, the problem at hand is transport dominated, and instabilities occur whenever a non-negligible fluid velocity is considered. In [29], Dawson stabilized the transport problem, while at the same time suppressing the unphysical upstream waves, using upwind finite differences for the approximation of the second order derivative in (2.13). Although this strategy proved to be successful, it presents two main drawbacks: i) it introduces a considerable amount of numerical dissipation and, more importantly, ii) it forces the use of structured meshes which do not allow for local refinement strategies.

The Streamline Upwind Petrov Galerkin (SUPG) method makes it possible to maintain the accuracy of the L^2 projection approach while evaluating the second order derivative in the linearized free surface boundary condition. This is a powerful stabilization technique which has been applied in a variety of frameworks, however its application to meshless methods [119] and boundary elements [77, 78] is quite recent. A FEM-SUPG has already been coupled to a BEM problem in [52], in the framework of magnetohydrodynamic flows through a circular pipe. In that scenario a BEM technique is applied to recover the magnetic and velocity field outside the pipe while the FEM-SUPG framework is used to compute the fields inside the pipe. In the present work we used the FEM-SUPG technique to properly project the boundary conditions for the boundary element technique. In particular, the SUPG stabilization is applied in a domain of codimension one. In the framework of SUPG stabilization [2], we modify the generic shape function $H_i(\mathbf{x})$, used for finite elements approximation, in the following way

$$H_i^{SUPG}(\mathbf{x}) = H_i(\mathbf{x}) + \delta h \nabla_s H_i(\mathbf{x}) \cdot \frac{\mathbf{v}(\mathbf{x})}{\|\mathbf{v}(\mathbf{x})\|}. \quad (2.27)$$

Here the constant $0 \leq \delta \leq 1$ sets the local amount of upwind stabilization on each cell. Since we work only on the boundary of our domain, the full gradient is replaced by the surface gradient ∇_s . The results in the present work are obtained considering $\delta = 1/\sqrt{2}$. As we worked with roughly square cells, and h represents the cell diameter, δh represents a good approximation of the streamwise cell dimension. Unfortunately, since the flow velocity is the main unknown of our simulations, the local velocity direction is not available at the time of the matrix assembling. However, the dominant component of the flow velocity is given by \mathbf{v}_∞ and we can write the SUPG correction in terms of the asymptotic flow direction, namely

$$H_i^{SUPG}(\mathbf{x}) = H_i(\mathbf{x}) + \frac{1}{\sqrt{2}} h \nabla_s H_i(\mathbf{x}) \cdot \frac{\mathbf{v}_\infty}{\|\mathbf{v}_\infty\|}. \quad (2.28)$$

This SUPG strategy is used in equations (2.18) and (2.20) to modify the test functions ψ_i, ω_a into ψ_i^{SUPG} and ω_a^{SUPG} . In previous works, like [29] and [98], the second derivative in (2.16) is obtained directly through a finite difference scheme, requiring a structured grid and introducing consistency errors typical of finite differences. Since we make use of the variable $\hat{\beta}$ to compute such second derivative we have the possibility to enforce the SUPG stabilization two times separately, *i.e.* once for each derivation along x . The most important advantage given by the use of the weak formulation combined with the SUPG stabilization is that it allows the use of unstructured grids. In particular, it is possible to employ a free surface grid which is refined only near the body, or to

adopt a local refinement strategy based on *a posteriori* error estimates. In addition, SUPG is a strongly consistent method and does not introduce numerical dissipation in the system.

2.2 Numerical validation

To validate the behavior of our method in the treatment of free surface flows, two different problems are considered. In the first test case we treat the motion of a fully submerged prolate spheroid at constant speed. In the second problem we consider the stationary motion of a surface piercing body, the Wigley hull.

These flows are common benchmarks in the naval literature, so there are many available references to evaluate the accuracy of the method we developed.

2.2.1 Submerged prolate spheroid

The purpose of this section is to study the flow field past a fully submerged prolate spheroid advancing at constant speed in calm water. The domain we have considered is a box of fluid surrounding the spheroid. The longest axis of the spheroid is oriented along the x axis of the global frame of reference. The spheroid has been placed at a prescribed depth, intended as the distance between the main axis and the free surface. The free surface portion considered is represented by the upper face of the parallelepiped. The truncation faces of the domain, $\Gamma_{in} \cup \Gamma_{out} \cup \Gamma_{tank}$ in Fig. 2.3 are placed far enough from the spheroid, so as not to influence the solution near it. The considered prolate spheroid has an axis of length 5m along the x direction, of length 1 meter along y direction, axis of length 1 meter along z direction. So we compute $L_{sph} = 10\text{m}$.

The domain extends for $L_{\infty x} = 15 \times L_{sph}$ along the x axis, for $L_{\infty y} = 15 \times L_{sph}$ along the y and for $L_{\infty z} = 15 \times L_{sph}$ along the z axis. We chose a sufficient depth of the spheroid main axis in order to avoid the presence of strong nonlinear effects. A good depth is $d = 1.25 \times 4\text{m}$ where 4m is the diameter of the spheroid.

The overall height of the basin is enough to consider an infinite depth approximation. This will make possible a comparison between our data and the theory introduced by Havelock in 1931, and reported in [55], which will be used here as a benchmark. Havelock derived an analytical expression for the the wave resistance of a spheroid submerged in water.

The mesh on the spheroid is presented in Fig. 2.4. The grid is composed of 24 nodes along the longitudinal axis and of 4 nodes along the other two circular section. The free surface mesh is refined adaptively during the simulations, while the spheroid mesh is fixed.

Qualitative analysis of the wave pattern

We first want to highlight the importance of the SUPG method in the approximation of the second derivative in the linearized free surface boundary condition(see section 2.1.2). As example, we have chosen the flux with $U_{\infty} = 6\text{m/sec}$ and we show the isolines of the wave elevation, both when SUPG is used and not used in Fig. 2.5. Looking at the plots we can appreciate that the SUPG stabilization plays a key role in our model. As suggested in [29, 107] the addition of upwind terms in eq. (2.1c) suppresses

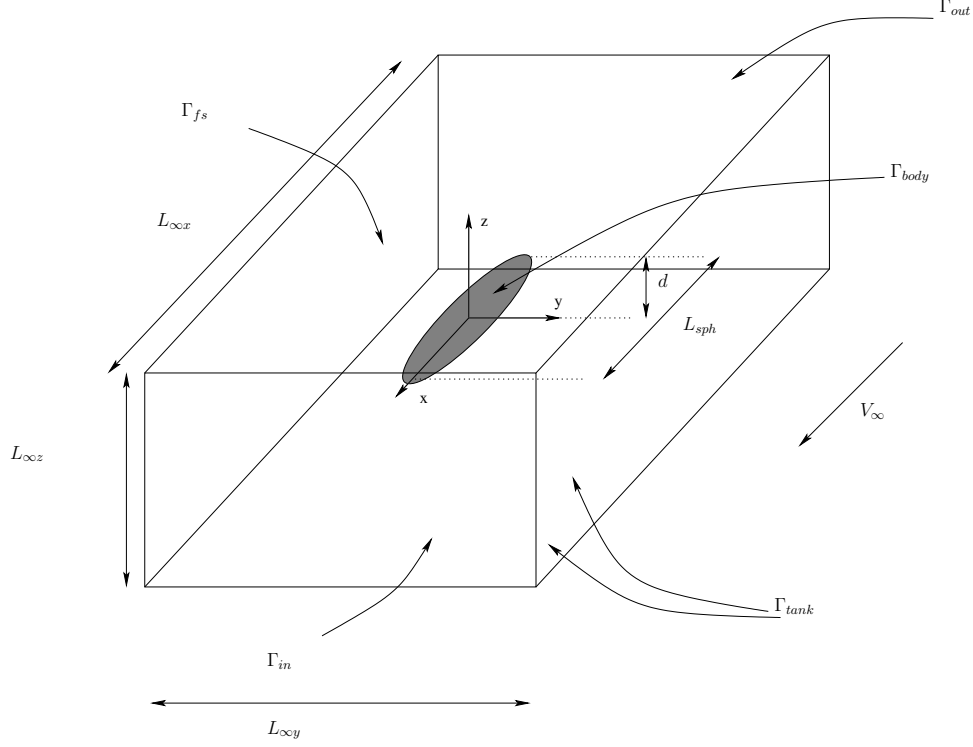


Figure 2.3: The spheroid is put under the free surface at a defined depth and it is possible to see also the outer tank set with $L_{\infty x}$, $L_{\infty y}$, $L_{\infty z}$. The flow enters from Γ_{in} , Γ_{tank} are the lateral surfaces and the bottom. The flow is considered at velocity V_{∞} directed as the x axis.

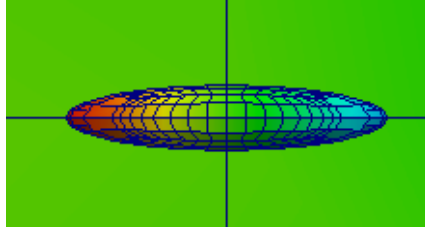


Figure 2.4: The fixed mesh on the prolate submerged spheroid.

the non physical upward propagating waves. In facts, we clearly see that without the SUPG there is a strong presence of waves which propagates upstream and any physical meaning of the solution is lost. Instead if we consider a SUPG stabilization [2], the V-shaped Kelvin wake pattern is recovered. The SUPG stabilization is a strongly consistent way of introducing upwinding in the linearized free surface condition. This is a first advantage because we introduce, using a strongly consistent method, less numerical dissipation in respect to a consistent method as an upwind finite difference stabilization. Even more importantly the SUPG strategy allows us not to use structured grids. The purpose of the present section is to see whether the developed method leads to solutions that are consistent with what can be found in literature about the waves created by a submerged prolate spheroid.

In Fig. 2.6 we present contour plots of water elevation fields computed at different velocities, and at the same depth. We see that both elevation fields present clear V-shaped

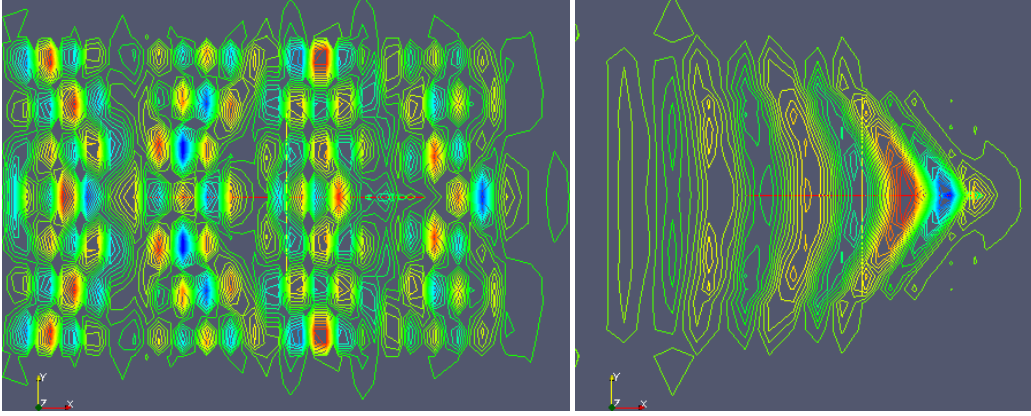


Figure 2.5: On the left, isolines representation of free surface elevation field for the chosen velocity $U_\infty = 2\text{m/sec}$ without the SUPG stabilization. On the right, isolines representation of free surface elevation for $U_\infty = 2\text{m/sec}$ computed making use of the SUPG stabilization. The colors on the contour lines represent the free surface elevation.

Kelvin wake patterns. Each wake is composed by a series of elevation peaks located on lines inclined of about 20° with respect to the main stream axis. Between the two arms of the V-shaped pattern, we observe a series of transverse waves. As expected, increasing the velocity of the flow, we observe a growth of the wavelengths of both V-shaped and transverse waves. These two peculiarities may be more or less evident depending on the velocity of the spheroid, which in this section is commonly expressed in terms of Froude number, defined as

$$Fr = \frac{U_\infty}{\sqrt{gL_{\text{spheroid}}}}. \quad (2.29)$$

The wave angle has been indicated in Fig. 2.6 for $Fr = 0.5$ and for the $Fr = 0.7$, to confirm that the V-shape angle behaves consistently with the simple model described in [26]. At $Fr = 0.5$, an angle $\alpha = 16.11^\circ$ is observed, while at $Fr = 0.7$ the angle

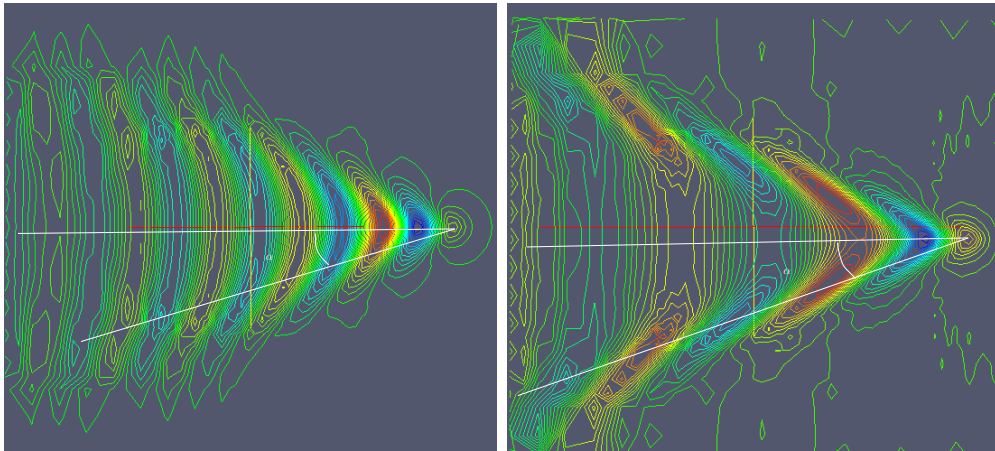


Figure 2.6: On the left, isolines representation of free surface elevation field for $Fr = 0.5$. On the right, isolines representation of free surface elevation for $Fr = 0.7$

is of 18.02° . This fact suggests that the developed method is able to reproduce one of

the most important features of a Kelvin wake. We see also that for the lowest Froude number there is a clear presence of a transverse system which becomes less important at high Froude numbers, and this is well recovered. At $Fr = 0.7$ we can see that the peaks, called featherlets, are aligned on the V-shaped Kelvin wake.

Comparison with literature results

We extract the maximum absolute displacement of the free surface elevation from the reference value $z = \eta = 0$. These values are compared to the ones reported in [107]. In his work the authors employed a nonlinear steady potential model. Even if the present method is based upon a linearized potential model, a comparison is still interesting. The curves of the wave elevation peak as a function of Fr number are presented in Fig. 2.7

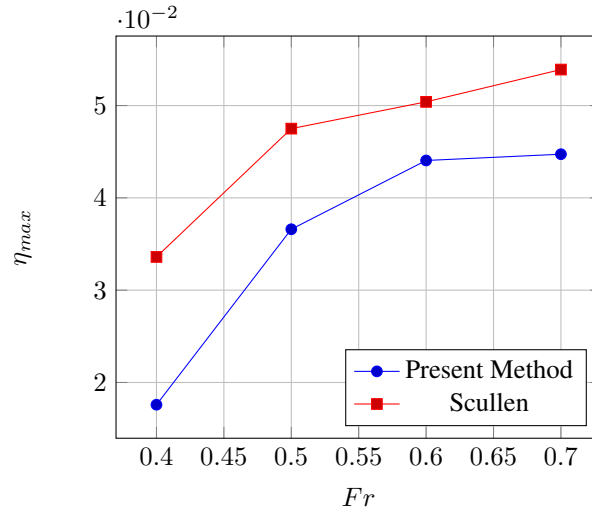


Figure 2.7: Maximum free surface elevation as a function of Froude number. The red line represents the results obtained using a non linear potential model, [107]. The blue line represent the results obtained with the present linearized potential method

The plot shows that the expected behavior is recovered by the model developed. Yet our linear method underestimates the elevation obtained with the more general and accurate nonlinear method. We stress that the present method underestimate the results reported in [107] since we have used a simple linearized free surface condition and not the fully non linear one. This would be consistent with an underestimation of the wave drag on the spheroid. It is therefore interesting to analyze the drag induced by the wave generation. In [55] the authors managed to compute an analytical formula to predict the drag of a submerged spheroid. To obtain a closed form solution, the spheroid was approximated by a series of doublets with a uniform volume distribution and their axis parallel to the spheroid main axis which, in the present work, is the same direction as that of the external flow velocity. Calling the major semiaxis length a and the minor

semiaxis length b :

$$e_c = \sqrt{1 - b^2/a^2} \quad (2.30)$$

$$A^{-1} = \frac{4e_c}{1 - e_c^2} - 2 \log \frac{1 + e_c}{1 - e_c} \quad (2.31)$$

$$D_w = 128\pi^2 g \rho a^3 e_c^3 A^2 e^{-p} \int_0^\infty e^{-pt^2} J_{3/2}(k_0 a e_c \sqrt{1 + t^2}) dt \quad (2.32)$$

Where $J_{3/2}$ is the Bessel function with base $3/2$, $p = 2gf/u^2$, $k_0 = g/U_\infty$ and f is the depth of the spheroid major axis. Considering a spheroid with $f = 0.25$ it is possible to compute the drag coefficient predicted by this theory for the present test case. Introducing the spheroid length L and its diameter d , the drag coefficient is defined as.

$$C_w = \frac{D_w}{\pi/6 \rho g L d^2} \quad (2.33)$$

Fig. 2.8 displays a comparison of wave drag coefficients between Havelock theory and the present method, for several values of Froude numbers (or of flow velocity). Recalling equation (2.5) we compute the pressure on the body as:

$$P = P_\infty + \frac{1}{2}\rho(U_\infty^2 - U^2) - \rho g z,$$

where z states for the height of the point in the considered framework. The drag coefficient can be obtained as the integral of the pressure coefficient as:

$$C_p = \frac{P - P_\infty}{1/2 \rho U_\infty^2}$$

$$C_w = \frac{1/2 \rho U_\infty^2}{\pi/6 \rho g L d^2} \int_S C_p dS$$

Again, the two curves show a similar behavior. The linearized model appears however to underestimate the drag, with respect to the analytical result. This error seems to be dependent on the Froude number, since it increases with the flow velocity. The same kind of error pattern has been observed also for different values of the depth f . This problem appears to be linked with what pointed out in Fig. 2.7. Since the our linearized model underestimates the height of the waves, it also underestimates the energy dissipated in the wave creation process, which leads to an underestimation of the drag force. The solution in [55] represents the theoretical solution and in order to recover it better we should use a non linear method as the one presented in [107]. We stress that our linearized method agrees with the theoretical result when the Froude number tends to zero. This is expected since the linearization is obtained considering an undisturbed reference configuration, like the one we would obtain with Froude equal to zero.

In the present work the wave coefficient only depends on the Froude number, as reported in [85] (page 31), in agreement with the so-called *Froude's hypothesis*. Such hypothesis states that the wave drag coefficient only depends on the Froude number, independently from the Reynolds number. *Froude's hypothesis* is confirmed by experimental data only when we are in presence of a thin boundary layer (theoretically of

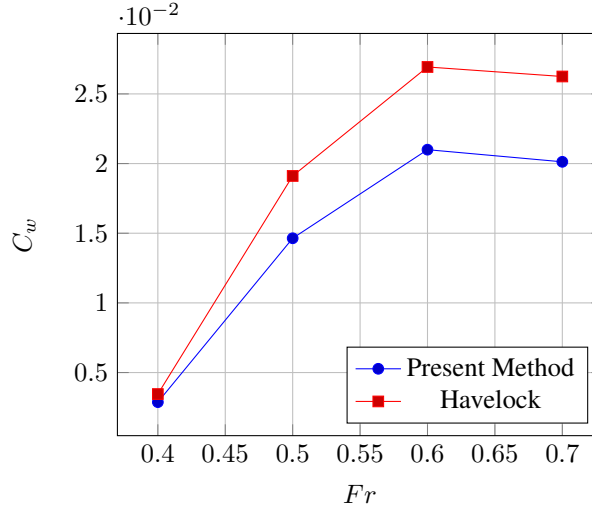


Figure 2.8: Wave drag coefficients of a fully submerged body as function of Froude number. The red line represents the analytical results as predicted by Havelock theory, [55]. The blue line represents the results of the present linearized potential model.

negligible thickness). This is verified especially for very big hulls, when the Reynolds number $Re = \rho U_\infty L_{boat} / \mu$ is higher than 10^5 , where μ is the viscosity of the fluid. For our simulation, considered $L_{spheroid} = 10$ meters we have Re values between $3.5 \cdot 10^5$ and $5.7 \cdot 10^5$ so we respect the *Froude's hypothesis*.

Quadratic BEM

As a comparison with Higher Order BEM, we present the results obtained using the same spheroid presented in the previous section at Froude number $Fr = 0.7$ when the finite dimensional spaces are both quadratic.

Quadratic elements should be able to better approximate the derivative of the unknown ϕ appearing in the linearized free surface boundary condition. We have chosen to use quadratic continuous elements for ϕ , $\frac{\partial \phi}{\partial n}$ and $\partial \phi / \partial x$. We note that in order to obtain a meaningful results with quadratic elements we have used the well known Telles quadrature formula [114].

We report the final mesh after 4 local refinements. We highlight that in Fig. 2.9

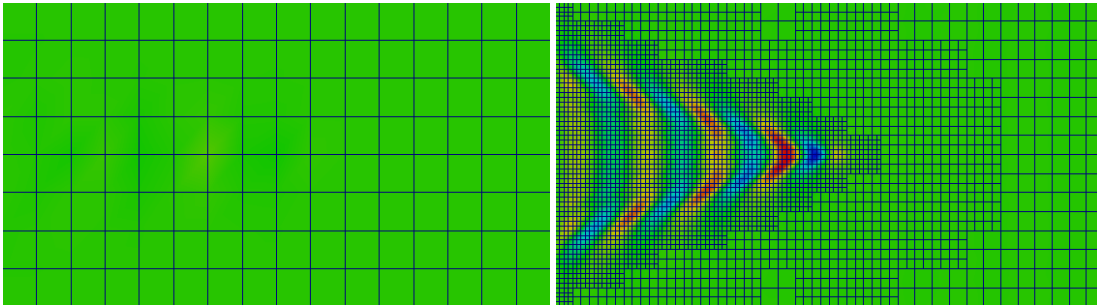


Figure 2.9: On the left the initial mesh. On the right the final mesh obtained in the submerged spheroid test case after 4 refinement cycles. Every 4 neighboring cells compose a single quadratic cell

every four cells represent one quadratic cell.

We report the comparison between what we have obtained in the linear case and this results with the quadratic, still continuous, BEM.

Table 2.1: Comparison between the quadrature formulas in the quadratic case at $Fr = 0.7$.

	Final C_w	Max wave height
LinearBEM	0.02099683	0.04405
QuadraticBEM	0.02120950	0.04436

The overall number of degrees of freedom is comparable between the two cases. So we have considered a less refined mesh for the higher order approximation. We see that, as we expected, the quadratic leads to a better approximation of the problem since the drag coefficient is increased and the wave pattern is better recovered.

2.2.2 Wigley hull

This section will describe how our model behaves in presence of a surface piercing body. The geometry considered for this test case, is that of Wigley hull. Given its simple analytic shape the Wigley hull is in fact a commonly used benchmark in hydrodynamics simulations, and several experimental data for such geometry are available in the literature.

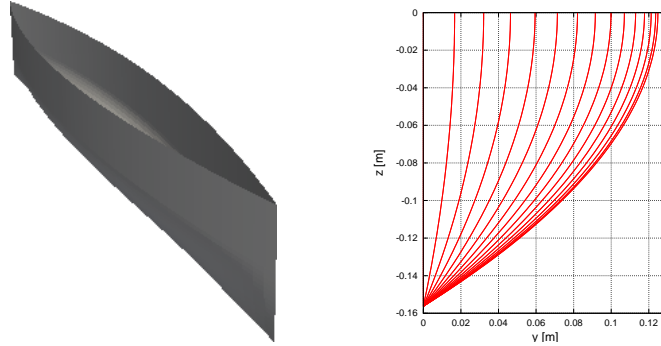


Figure 2.10: On the left, a three dimensional view of the Wigley hull used for the simulations. On the right vertical sections of the Wigley hull used for the simulations, generated by planes normal to the longitudinal axis of the hull.

The Wigley Hull is analytically described by the following equation

$$y(x, z) = \frac{B}{2} \left[1 - \left(\frac{2x}{L} \right)^2 \right] \left[1 - \left(\frac{z}{T} \right)^2 \right] (x, y) \in [-1, 1]^2. \quad (2.34)$$

Where y is the span of the hull, x its length and z its depth; in the present calculations the hull length used is $L_{boat} = 2.5\text{m}$, the span of the whole hull $B = 0.25\text{m}$, and its total depth beneath the undisturbed free surface is $T = 0.15625\text{m}$. Fig. 2.11 shows a sketch of the Wigley hull test case domain.

The domain is extremely similar to that used for the submerged spheroid. The numerical tank has dimensions $L_{\infty x} \times L_{\infty y} \times L_{\infty z}$. While the free surface boundary Γ_{fs}

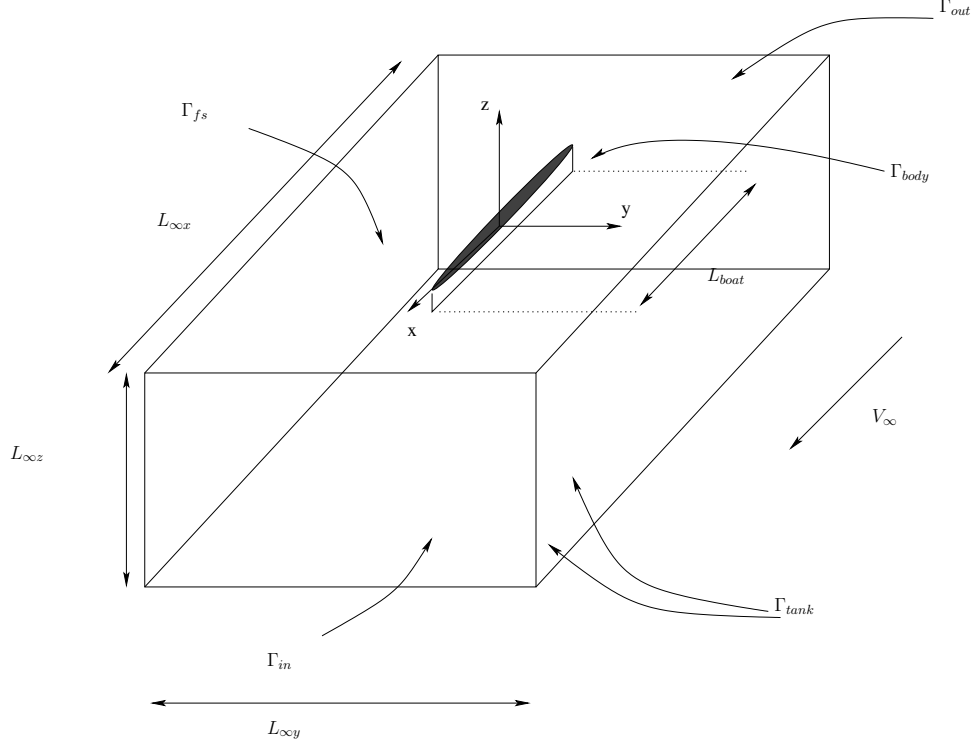


Figure 2.11: A sketch of the numerical domain. The wigley hull is placed in the center of the free surface. In the picture it is possible to see also the outer tank set with $L_{\infty x}, L_{\infty y}, L_{\infty z}$. The fluid enters from Γ_{in} , Γ_{tank} are the lateral surfaces and the bottom

is located at $z = 0$. The hull is located at the center of Γ_{fs} .

This geometrical configuration coincides with that of a set of experiments performed at the university of Tokyo [74], which will be used as a benchmark in this test case. Six different velocities will be tested in order to compare them with the experimental results in [74]. The Froude number is defined again as

$$Fr = \frac{U_{\infty}}{\sqrt{gL_{boat}}} \quad (2.35)$$

Local refinement strategy and flow inspection

We have employed the same local refinement strategy already described in the submerged spheroid test case. On the boat, the grid is composed of 4 cells along the height of the hull, and 32 cells along its length. On the free surface, the mesh is set to be more refined around the hull and, to concentrate the degrees of freedom where the waves are expected to be.

We have computed the results using 5 local refinements on the free surface. We see the initial and final mesh for one of the cases considered in Fig. 2.12: For the Wigley hull at two of the Froude numbers considered in the local refinement tests, we perform a qualitative comparison with the simple model described in [26].

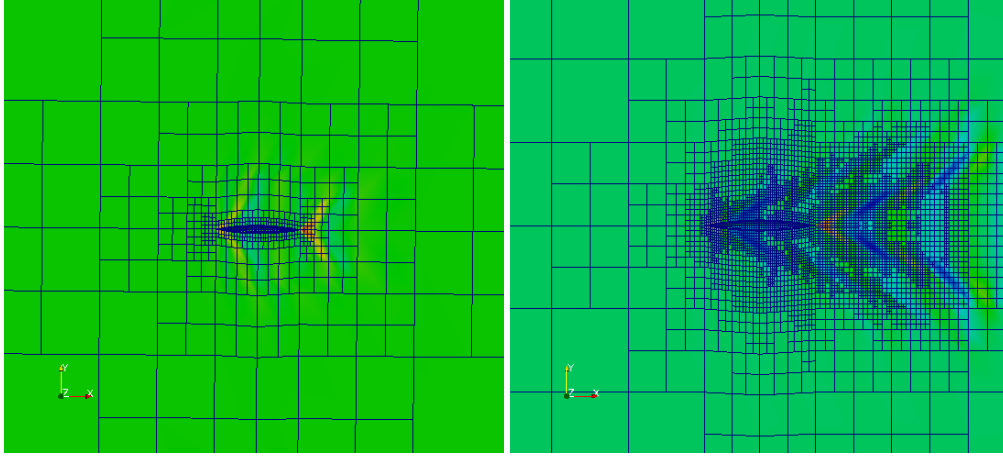


Figure 2.12: On the left, the free surface mesh at the initial level of refinement. On the right, the free surface mesh obtained after five refinement cycles

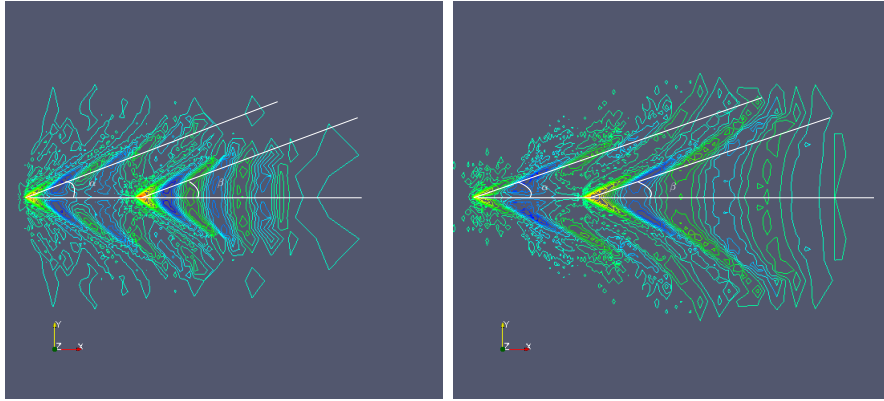


Figure 2.13: On the left, the Kelvin wave angle for $Fr = 0.267$ and on the right, the Kelvin wave angle for $Fr = 0.354$

Table 2.2: Kelvin angle for the Wigley hull

Fr	angle α	angle β
0.267	21°	20°
0.354	19°	18°

The two angles remain almost constant varying the Froude of the simulation. This fact agrees with what is reported in [26], and confirms that the present method is able to reproduce one of the most important feature of a Kelvin wake.

The local refinement strategy, since we are using an *a posteriori* error estimator, has a disadvantage in respect to what can be achieved using a prebuilt fixed mesh. This strategy needs to solve many linear system to refine the grid where the gradients of the unknowns are the highest. It also uses a large number of cell in its latest cycle. This situation increases the computational costs of the present method, the results of the local refinement strategy have been obtained in about 8 hours each. The major advantage is the possibility to concentrate the degrees of freedom only where the solution is more rapidly changing. This leads to better results.

Comparisons between the computed profiles on the hull surface, and corresponding experimental results of the University of Tokyo are now presented in Fig. 2.14 The

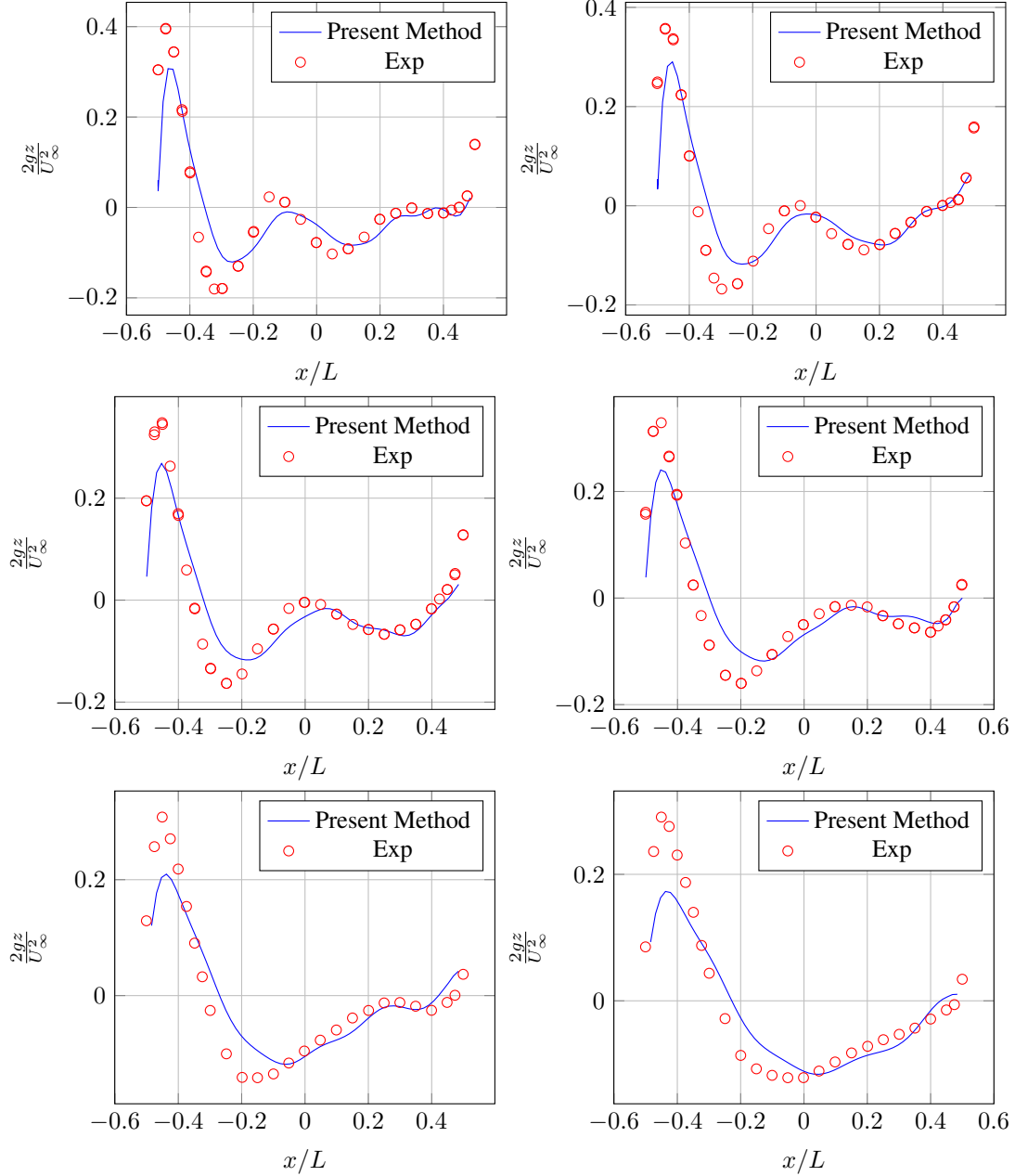


Figure 2.14: Non dimensional free surface elevation $2gz/U_\infty^2$ on the Wigley hull surface as a function of non dimensional longitudinal coordinate x/L , at different Froude numbers $Fr = 0.250, 0.267, 0.289, 0.316, 0.354, 0.408$. The blue continuous line represents the result of the present method. The dots represent the university of Tokyo measurements [74]

plots represent a comparison between experimental and computed wave heights along the hull, for $Fr = 0.250, 0.267, 0.289, 0.316, 0.354, 0.408$. In agreement with the experimental results the positive peaks on the hull are located just after the bow and near the stern. The plots suggest that, from a quantitative point of view, the model repro-

duces rather correctly the horizontal position of the peaks. Also the wavelength along the hull is correctly reproduced. As the Froude number increases the experimental water wave wavelength increases. The present method is able to reproduce such behavior too, as the location of the peaks remains quantitatively correct for all the Froude numbers considered. Despite this, the present model clearly underestimates the wave elevation. The problem is probably due to the choice of linearized free surface conditions. A linearized model is in fact not able to fully represent the wave generation process, as suggested in [98]. The results of Fig. 2.14 have been obtained using a linear continuous approximation for all the three unknowns ϕ , $\partial\phi/\partial x$ and $\partial\phi/\partial n$. For the last unknown a particular strategy was used to recover properly the derivative values on nodes belonging to both water and hull (see [48,49]). Without such treatment, the computed solution results extremely inaccurate, since the normal derivative approximation is an average of the values on the boat and on the free surface, which are extremely different.

Comparison with other models

In this section we want to compare the results of our linearized potential model proposed with those obtained with different models. In the first case (see Fig. 2.15 on the left), we compare the present model solution with the established results of other linearized methods, [98]. In a second case (see Fig. 2.15 on the right), the present model will be compared to that obtained with an unsteady potential model with fully non linear free surface boundary conditions, presented in [78]. For both cases the test case considered is that of the Wigley Hull at $Fr = 0.316$.

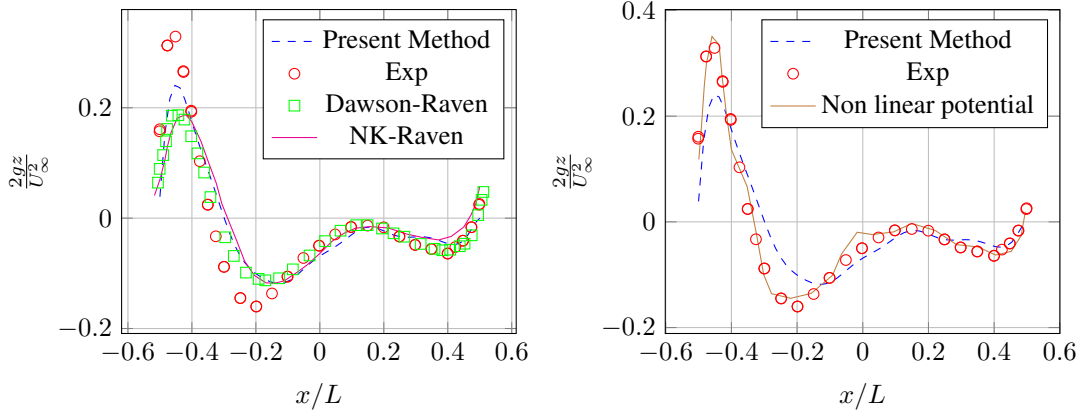


Figure 2.15: Non dimensional free surface elevation $2gz/U_\infty^2$ on the Wigley hull surface as a function of non dimensional longitudinal coordinate x/L , at Froude number 0.316. On the left we compare the developed method with other linearized models. The blue continuous line represents the waterline obtained in this work. The red dots represent the experimental results as reported by Ikemata et al. in [74]. The green line represent the results of the Dawson method, [98]. The magenta line represents the result of the Neumann–Kelvin method, implemented in [98]. On the right we compare our method with a non linear BEM. The blue continuous line represents the waterline obtained with our BEM. The red dots represent the experimental results [74]. The brown line represents the result obtained using a non linear BEM presented in [78].

We first consider the plot showing the comparison among the wave profiles on the

Wigley hull obtained with the linearized models. As we can see all the linearized methods considered underestimate the height of the peaks in the water profile on the hull. Thus, it was correct to speculate that the underestimation of wave elevation was mainly due to the choice of a linearized boundary condition on the free surface.

It is interesting to point out that the present model has less numerical dissipation than the other linearized strategies, as it employs the strongly consistent SUPG stabilization method. As a result, we see that the wave crest is roughly 30% higher with respect to other linearized methods.

From the hull wave profile plot on the right, we can appreciate that the non linear potential model recovers the water elevations in a significantly more accurate fashion. This confirms one more time that the error of our method is mainly due to the free surface linearized model. Nonetheless we want to assess if the error can be further reduced by the choice of different boundary elements.

Quadratic BEM

Since we have proved the possibility of using a quadratic BEM in the submerged spheroid test case we now see its application on the Wigley hull.

In this case the free surface interacts directly with the body since it pierces the free surface. We remind that we are using an isoparametric BEM so we use the same quadratic approximation even for the geometry. This should increase the accuracy of the method because the hull has got a parabolic, therefore quadratic, formulation.

For this comparison we use $Fr = 0.316$ as we have already done in section 2.2.2. We have used a less refined mesh in the quadratic simulation. Thus the computational

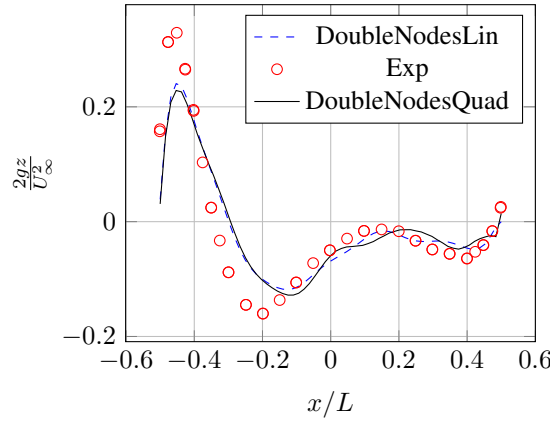


Figure 2.16: Non dimensional free surface elevation $2gz/U_\infty^2$ on the Wigley hull surface as a function of non dimensional longitudinal coordinate, at Froude number 0.316. The blue continuous line represents the waterline obtained with our linear BEM. The red dots represent the experimental results as reported in [74]. The black continuous line represents the waterline obtained with our quadratic BEM.

time is the same in both the simulation. We see that the two waterline almost overlap. Nevertheless the possibility of having non linear BEM approximation is very important. The calculation presented in the present chapter state that the quadratic BEM is able to recover the well known results both for the submerged spheroid and the Wigley hull.

CHAPTER 3

Parallel implementation of a Boundary Element Method and its acceleration using a Fast Multipole Method

In Chapter 1 and 2 we derived a BEM for the Laplace equation and we applied it to the study of ship wave interactions. We exploited the related Boundary Integral Representation to express the solution at each point of the domain in terms of convolutions on the domain boundaries between a fundamental solution and the boundary trace of the solution and of its normal gradient. The BEM induces a computational cost of order $O(N^2)$, and the numerical solution of BEM problems is far from trivial. If one chooses to assemble the full matrix on a regular computer/laptop/workstation, the number of degrees of freedom is roughly limited to $O(10^4)$, mainly due to the computational effort required to assemble and store the $O(10^8)$ elements of the system matrix. Increasing the number of unknowns makes the assembly of such matrices almost unbearable also from a memory point of view. Storing a full matrix for 40 thousands double precision unknowns already requires more than 10 GB in RAM memory, which is close to the limit of user-level desktops.

In Chapter 1 we have seen some numerical strategies (high order elements, local refinement strategies) to reduce the number of unknowns in BEMs. However, as the size of problems increases, none of these techniques alone is enough to reduce memory problems. Splitting of the problem into subdomains and distributing its execution over multiple CPUs is one possible option, which is also beneficial for wall-time computational costs. Domain decomposition can be achieved at the partial differential equation level, or at the algebraic level. Each approach has its own advantages and disadvantages, but both methods increase the limit on the degrees of freedoms by exploiting distributed memory techniques, and reducing the amount of degrees of freedom han-

dled by each single processor.

In recent years many library have been developed to address the BEM requirements. In [111] and [94] the authors present an effective implementation of high order boundary element methods. In [111] the boundary element method is parallelized using a shared memory parallelization and the authors focus on the implementation of different kernels using high order methods, while in [94] the authors present a hybrid parallelization scheme and the combination of high order methods with adaptive quadrature formulas. To the best of our knowledge, no OpenSOURCE library is available in the literature that combines high order elements with local refinement strategies for boundary element methods on arbitrary geometries using HPC platforms.

In this chapter we present a new OpenSOURCE BEM library, called π -BEM, that gathers together several algorithms and ideas in a flexible, modular and extendible way, exploiting both shared and distributed memory parallelisms. We split the computational effort at the algebraic level for distributed parallelization, by combining a domain decomposition method based on the graph partitioning tool METIS [35], with the high performance computing library Trilinos [57] used to tackle distributed linear algebra. We use Intel Threading Building Block (TBB) [99, 115] to exploit multicore architectures. A similar combination has been successfully applied to achieve high computational efficiency in fluid dynamics, as demonstrated in ASPECT [63]. In our BEM library the distributed memory parallelism leads to very high performance benefits, due to the structure of the matrix assembling procedures.

In recent years many methods have been developed to approximate the action of a BEM matrix-vector product in order $O(N)$ operations instead of assembling the full matrix. Examples of such methods are the Fast Multiple Method [46] or Hierarchical matrices [43]. Successful BEM libraries and solvers should feature some or all of these expedients aimed at the reduction of the main BEM bottle-necks. Many high performance libraries take advantage of hierarchical matrices, or H matrix concept. For example BETL [58], is based on AHMED [14, 15]. Alternative acceleration methods are based on non uniform fast Fourier transforms [3], and we present a preliminary parallelization of the Sparse Cardinal Sine Decomposition (SCSD) in Chapter 4. For most BEM libraries the fast multiple method is the acceleration method of choice, as it guarantees very high reductions from the computational point of view [46]. Greengard proposed a pure multithreaded version in 1990 [45], and only recently Yokota *et al.* developed a parallel version of the algorithm using hybrid coding techniques [121]. Several works are dedicated to assessing the applicability of FMMs to exascale problems (see, e.g., [13] and the references therein). In many cases the panels coming from the boundary discretization are directly used in order to set up the FMM hierarchical space subdivision. This approach leads to some modification to the algorithm in order to guarantee that all mathematical assumptions are properly satisfied [94].

We have performed a preliminary acceleration test of the developed parallel BEM library π -BEM, using a FMM. Given the extreme flexibility and generality of the original BEM solver the coupling with an existing FMM library, as the one depicted in [121], is extremely difficult. Moreover, the relative small size of the problem (at most $O(N^6)$ unknowns) sets us in a different setting with respect to exascale libraries as the ones developed in [13], and we don't approximate near BEM interaction by means of dielectric models, or BIIBE, as in [122]. We developed our own implementation of FMM capa-

ble of dealing with the characteristics of π -BEM. High order methods, local refinement and double nodes handling bring a considerable increase in the algorithm complexity when compared to classical BEMs, making it mandatory to use hybrid parallelization techniques. A hybrid MPI-TBB parallelization strategy allows for an optimization of the various algorithms, while maintenance and ease of use is achieved by exploiting the `deal2lkit` library [102].

In Section 3.1.1 we analyze the computational cost of the BEM algorithm to highlight the real bottlenecks of the code, in Section 3.1.2 we describe an efficient automatic domain decomposition to balance the workload among different processors, then we analyze the performance of the parallelization using both a strong scaling, Section 3.1.3, and a weak scaling, Section 3.1.4, analysis. In Section 3.2 we present the acceleration of the method through a Fast Multiple Method, in Section 3.2.2 we present a validation of the BEM-FMM coupling against the standard BEM solver. Finally in Section 3.2.3 we discuss the parallelization of the accelerated BEM algorithm.

3.1 Parallel Boundary Element Method

3.1.1 Profiling

We report the time analysis of a test case scenario presenting all the computational challenges highlighted in Section 1.3:

- Laplace Boundary Element Method on the truncated pyramid test case, see Figure 3.1, considering 6534 degrees of freedom.
- Double Nodes strategy [49] to treat the presence of sharp edges.
- Mixed Neumann Dirichlet boundary condition.

The timings of the analysis are reported in Table 3.1. We see that the matrix assembly is the major part of the overall program, while the post process gradient recovery is almost a negligible cost compared to the rest. In the following we analyze with greater

Table 3.1: *Profiling of a direct BEM solver on a single CPU.*

Function	Time (sec)
Assemble cycle	68.44
Solve Time	5.741
Post Process (Gradient Recovery)	0.1645
Total Time	78.6

detail each function to spot possible parallelization strategies.

Matrix assembling Since this function is extremely time consuming we need to parallelize it very efficiently. We use a collocation scheme (see Section 1.2) to solve the boundary integral formulation represented by equation (1.5). Provided that the whole computational grid is available on each processor, every line of the matrix can be assembled independently, making this an embarrassingly parallel computation, where MPI is an optimal strategy for parallelization, and we expect a theoretically linear scalability.

The assembling can be sketched as follows:

1. Loop over all collocation point.
2. Integrate over the entire domain the coupling between each basis function and the given collocation point.

The first loop is split among all MPI processors, and every processor assembles a different slice of the overall system matrices in (1.9). To preserve the parallelization efficiency it is mandatory to achieve a proper workbalance between different processor. This is achieved with a graph partitioning tool to ensure that the number of unknowns is kept equal between different MPI processors, see Section 3.1.2.

Normal vector and gradient recovery This part of the code is needed to approximate the gradient of the solution, namely $\nabla\phi$ on the edges of the domain. This technique can be very useful for post-process purposes, see Section 1.3.5. We need to discretise, in a distributed memory environment, equation (1.16) which represents a standard L_2 projection in Finite Element Methods. This part does not present particular difficulties, and we followed standard procedures (see, for example, the tutorials of the `deal.II` library).

3.1.2 Domain decomposition and workbalance

Parallel approximations of boundary value problems require a splitting of the domain which is distributed among processors, and where each processor only knows about a part of the computational domain. This is a key ingredient to upscale Finite Element computations where the interactions are limited to the support of the basis function of the FiniteElement space, and where locality of the algorithm allows for computations which are independent across the processors, a part from a thin layer of overlap between different subdomains. In Boundary Element Methods, however, each matrix entry depends on information *on the entire boundary of the domain*, and while computations can be effectively parallelized, every processor still needs access to the full discretization of the boundary of the geometry. This is the main difference between FEMs, where the result is a big sparse matrix that can be computed by splitting the work almost independently on each processor, and BEMs where the result is a smaller dense matrix, where the computation of each entry requires information on the entire domain, that needs to be shared among all processors. We remark that since the computational domain is only the boundary of the geometrical domain, the need to share it across all processors is rarely a bottle-neck, while in general it is mandatory *not to share* the full domain in FEM computations.

The computational effort is split among processors by sharing the full triangulation on all processors, and marking each subdomains with different ids, which are then assigned to different processors in the assembly routines. We use the graph partitioning tool METIS [35] to split the domain into subdomains among processors, and some reordering of the degrees of freedom to maximize locality in distributed vectors. Figure 3.1 shows an example of the automatic domain decomposition of a locally refined grid we can achieve using the graph partitioning tool. We have considered 16 different processors and we have plotted the corresponding domain in different colors. We clearly see that in the case of locally refined grids the partition is not trivial.

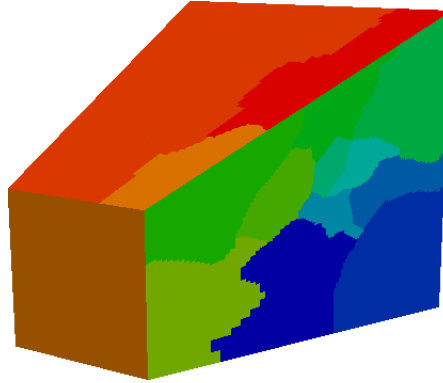


Figure 3.1: Domain decomposition between 16 different MPI processors. The grid has been locally refined and then split using the graph partitioning tool METIS. The different colours represent the areas concerning each different processor.

After we properly set up a domain decomposition ensuring that each processors takes care of a balanced number of unknowns we need to split the corresponding vectors and matrices. In recent years many libraries have been developed to deal with parallel linear algebra in a user-friendly way. The idea is to provide a high level interface to the user letting the internal implementation taking care of the actual parallel handling of the degrees of freedom (communication and range partition). For example both the PETSc library, [8], and the Trilinos project, [57, 63], provide this kind of interface. Of these two libraries, only Trilinos is currently thread-safe, and we exploit this feature to setup an efficient hybrid multiprocessor multithread parallelization on top of METIS partitioning.

The solution of the final (dense, non-symmetric) problem is obtained through a parallel implementation of the preconditioned Generalised Minimal RESiduum linear solver [101]. The preconditioner is obtained through an Incomplete Gauss factorization of a band matrix extracted from the system matrix for the standard BEM. Even in parallel computation this preconditioner has a reasonable memory and CPU footprint, and it is accurate enough to reduce sensibly the iterations required to obtain a solution.

3.1.3 BEM: strong scaling

We execute a scaling analysis up to 2 nodes, where each node is composed by two 10 cores (E5) Intel Xeon E5-2680 v2 for a total of 40 processors. However, our infiniband network drivers only allow for 16 MPI processor when we account for extranode communication. This may influence the scalability results when we exploit all 40 different processors, especially when the application does not require many in-node computations. We perform our first analysis on the same mixed Dirichlet-Neumann test case of Section 1.3. We impose the exact solution (1.17) and its corresponding normal derivative on the truncated pyramid domain depicted in Figure 1.6. We report the scalability analysis in Figure 3.2. The picture on the left represents the analysis using 5 global refinement, while the one on the right depicts the speedup using 6 refinement. The circled blue lines represent the overall speedup, the squared green plots are the speedups in the assemble cycle, the red lines with triangles show the speedup in the solver cycle,

Chapter 3. Parallel implementation of a Boundary Element Method and its acceleration using a Fast Multipole Method

and the cyan starred plots draw the post process scalability. We plot the ideal speedup as a reference using red circles.

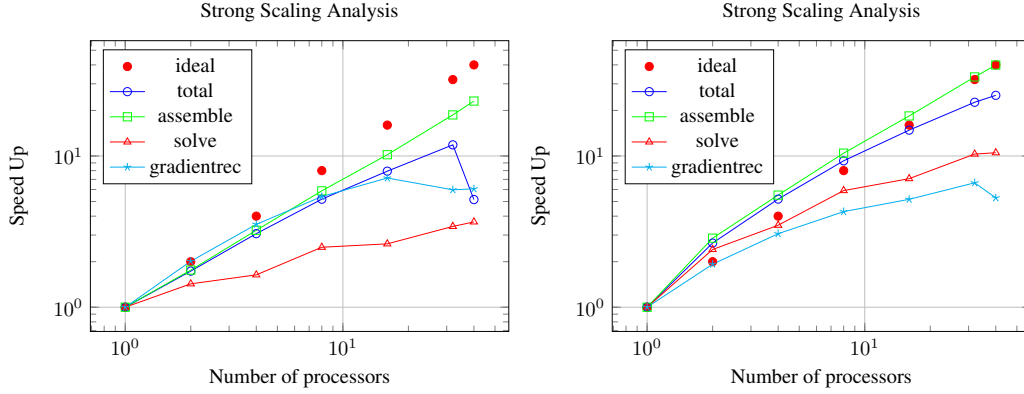


Figure 3.2: Strong Scalability test. The Figure on the left represent the analysis using 6534 degrees of freedom and the Figure on the right the analysis using 25350. We test up to 40 processors. We report the scaling considering the worst timing on all the used processors. In blu with circles we plot the total scalability, in green with squares the performance of the full matrix assembling, in cyan with stars the gradient recovery scalability, and in red with triangles the performance of the linear solver. The red dots represents the ideal speedup.

Looking at the left plot of Figure 3.2, we observe an almost linear behavior up to 4 processor in the overall scalability, and then we observe some non optimal behavior. In particular, the solving phase does not present optimal scalability, due to the communication overhead needed for each matrix vector multiplication. The only method that maintains a linear, nevertheless suboptimal, behavior is the function that takes care of the assemblage of the matrix, where no communication is involved. The normal and gradient recovery function has clearly a suboptimal behavior, again due to communication overheads in the assembling cycle of the sparse system for the gradient recovery (1.16). We believe that the communication overhead with 6534 dofs is the reason for the loss of performance between 32 and 40 processors. A possible explanation to the overall performance loss lies in the non optimal communication setting of the infiniband driver between different nodes. This is very relevant in the considered case since we only require 6534 degrees of freedom. The right plot of Figure 3.2 confirms this analysis. Increasing the number of degrees of freedom to 25350, we see that the code has almost an optimal scalability up to 8 processor (the slight super-optimality is due to some cache effects passing from 1 to two processors). When the number of degrees of freedom increase, the communication overhead becomes negligible and it no longer induces a performance loss. The least optimal function is the one that takes care of the parallel resolution of the system. This is expected since it is mainly based on many matrix vector multiplication. Each Matrix vector multiplication implies a little communication overhead to properly set up the operation in a MPI environment.

For reference, we plot the relative importance of the assembling VS solving cycles in Figure 3.3, where we consider the analysis with 25350 unknowns and we plot in blue with circles the time fraction needed by the assemble cycle and in green with square the percentage needed by the solving cycle. Increasing the number of processors we expect the assembling function to keep behaving almost optimally. The increasing

number of iterations required by the linear solver introduces some new communication overhead causing the corresponding function to scale sub-optimally and becoming another bottleneck of the overall algorithm. The simple ILU preconditioner has a sub-optimal behavior since the number of iterations required is highly depending on the number of processor. A possible solution to this issue lies in the implementation of a more effective preconditioner. An in-depth study of MultiGrid algorithms for mixed Dirichlet-Neumann problems is currently undergoing.

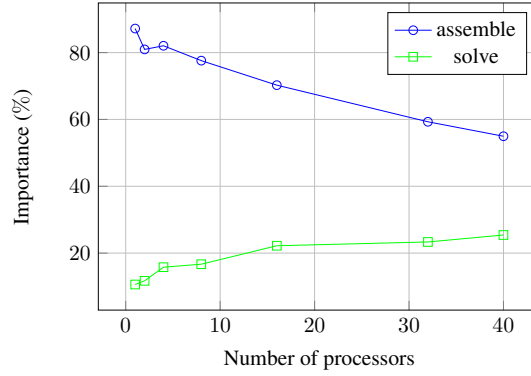


Figure 3.3: The relative importance, in terms of computational time, of the two main methods of the BEM code. Simulation up to 25350 unknowns. The blue circled lines represents the assembling of the full matrices, the green squared the importance of the linear solver.

3.1.4 BEM: weak scaling

To complete the analysis of the current parallelization strategy we analyze the weak scaling behavior of π -BEM. Since the computational cost of a standard Boundary Element Method is quadratic, order $O(N^2)$, we consider a number of processors that goes as the square of the number of unknowns, to maintain the number of matrix entries roughly fixed on each processor. Such an analysis points out the efficiency of the code if we fix the workload per processor. Ideally we would expect the time to remain constant. We performed our analysis up to 256 MPI processors and 25350 degrees of freedom. We report the weak scalability analysis in Figure 3.4. We plot the ratio between the timings on a single MPI processor and 1734 degrees of freedom and the actual timing required. The circled blue line represents the overall performance, the squared green plot is the timing of the assemble cycle, the red lines with triangles show the performance in the solver cycle and the cyan starred plot draws the post process timings. We plot the ideal performance as a reference using red circles. The assembling routine has a superoptimal behavior. This is probably due to the structure of the assembling cycle. While we guarantee that the number of matrix elements are the same on each processor, each processor has to loop over all the cells N times less. This should explain the superoptimal behavior. The matrix assembling is the leading term for the computational cost and induces the superoptimality up to 16 processors. If we increase the number of MPI processors to 256 we increase dramatically the computational costs for the linear solver, explaining the suboptimal behaviour of the solving phase, as we see in Figure 3.3.

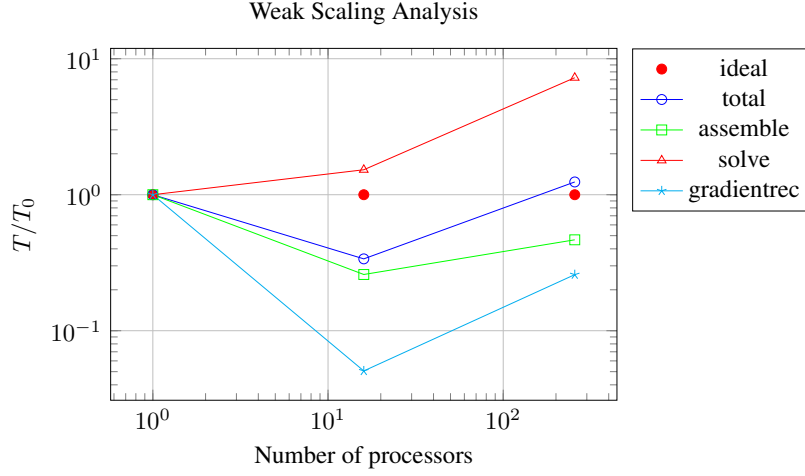


Figure 3.4: Weak Scalability analysis up to 25350 degrees of freedom and 256 processors. We report the performance considering the worst timing on all the used processors. In blue with circles we plot the total performance, in green with squares the timing ratio of the full matrix assembling, in cyan with stars the gradient recovery performance, and in red with triangles the performance of the linear solver. The red dots represents the ideal timings.

3.2 Accelerated Boundary Element Method

In Chapter 1 we showed how a reliable and accurate BEM solver can be improved by higher order elements, local refinement, and accurate capturing of geometry features. We have also proved how a proper parallelization strategy in a distributed memory system (MPI) can effectively improve the computational efficiency of the method. However, the cost for the solution of standard BEMs remains quadratic, due to the dense structure of their system matrices. This fact poses obvious limitations to the number of degrees of freedom that can be considered, both in terms of memory requirements and computational times. High order elements and local refinement strategies mitigate this issue, but even in such frameworks, most problems of interest result in algebraic systems which prove unmanageable both in terms of computational and memory requirements. In π -BEM we integrated high-order elements and local adaptive refinement with well known methodologies to accelerate the BEM solver.

In the last decades many methods have been developed to reduce the computational cost from $O(N^2)$ to $O(N \log N)$ or even $O(N)$. Many of these method exploit the structure of the fundamental solution G to approximate the convolution integrals when two cells are well separated. Some examples are Hierarchical Matrices, [20], Sparse Cardinal Sine Decomposition, [3], and Fast Multipole Method, [45, 46].

The application of H Matrices and SCSD to the BEM discretization is currently under study. As for now, the π -BEM library has been complemented with an implementation of the Fast Multiple Method. Such Implementation has been guided by the same paradigms we adopted for the development of the direct BEM discretization. As shown in Section 1.3, we designed π -BEM to treat a vast variety of mixed Laplace problems, using Lagrangian basis functions of arbitrary order, and with local adaptive refinement strategies on complex geometries. The matching FMM algorithm should be able to automatically treat any geometrical layout, nodes distribution on the domain, and (pos-

sibly changing) quadrature formulas over several processors. These characteristics of π -BEM make it difficult to use existing OpenSOURCE FMM libraries, even though they have outstanding performance, like, for example, ExaFMM [122]. These libraries are optimized for simpler particle dynamics simulations based on N-body problems, and are very sensitive (and difficult to tune) when large anisotropies are present in the distribution of particles, or when specialized (and different) quadrature formulas (corresponding to varying particle distributions) have to be used in different parts of the domain. Our attempts to use such libraries in the context of π -BEM have so far failed, and we therefore provided our own implementation.

3.2.1 Fast Multipole Algorithm

We follow the extremely clear explanation provided by Greengard and Gropp in [45] to provide an informal description of the Fast Multipole Method. For a detailed mathematical description see [76]. The algorithm has originally been derived for N-body problems and it takes into account N evaluation points (nodes) and M charges (sources) that are distinct in space. N-body problems are typical in electromagnetic and astrophysics applications in which one is interested in computing the force exerted by the charges or masses in correspondence with the evaluation points. The computation is based upon pairwise evaluations of gravitational or electromagnetic potential which coincide with the Green functions considered in this work. This results in the evaluation of NM source-target distances. If the source and target points coincide we recover the quadratic cost $O(N^2)$. The Fast Multipole Method is based on the consideration that an harmonic expansion of the electromagnetic or gravitational potential of a single charge, allows the separation into distinct factors of the effects of the source and evaluation point respectively. In principle these factors can be computed separately only once for each source and node point leading to a $O(N)$ computational cost. In practice the harmonic series converges only when target and node points are well separated. Thus one important building block of the FMM is a hierarchical space subdivision which is used to decide whether harmonic expansions or direct potential evaluations are to be used. Other important ingredients of the algorithm are, the *multipole expansion*, and the *local expansion*, which basically sum the contribution of groups of sources on well separated groups of nodes. These instruments lead Greengard and Rokhlin to the development of a very efficient algorithm for the fast ($O(N)$) evaluation of the N-body problem interactions.

Observing that the electromagnetic or gravitational potentials of a single charge coincide with the free space Green functions employed in most BEM discretizations, it is possible to modify the FMM algorithm and integrate it into a BEM solver. Since the BIE integrals are computed numerically, a BEM discretization can be recast into a N-body problem in which the N nodes represent the collocation points and the M sources correspond to the quadrature points. The full evaluation of each BIE integral, which is equivalent to a matrix vector product in the direct BEM framework, corresponds to the solution of one N-body problem. More specifically, the integral from which matrix D in equation (1.9) is originated corresponds to a N-body problem for the gravitational potential while the integral from which matrix N is derived corresponds to a N-body problem for the gravitational force.

Even considering an arbitrary number of collocation points (high order Lagrangian

basis function), an arbitrary number of quadrature points per cell (arbitrary Gaussian quadrature formula) and a non uniform cell distribution over the domain (adaptive local refinement strategies) we still fall within such N-body FMM framework. Yet, all these complicated features of our specific BEM implementation require a flawless hierarchical space subdivision, and an equally flawless assessment of close and far FMM interactions. An algorithm unable to detect the correct type of interactions in all the possible geometric situations leads in fact to non converging harmonic series expansions and eventually to a FMM algorithm which does not converge to the BEM direct solution on all possible geometries.

Moreover, in Section 1.2 we highlighted the necessity of using specific quadrature formulas for the proper integration of the weakly singular integrals coming from the boundary integral formulation. On each line of the BEM matrix vector product a different set of source points, corresponding to the specific quadrature formulas considered, is moved in a different position and presents different intensities. In such conditions it is not possible to exploit an external FMM library without implementing massive changes in the way the direct short range interactions are computed and stored. We remark that the same problem is not as dramatic for BEM formulations characterized by very specific choices of shape functions (piecewise constant) and quadrature rules (middle point) or based on kernel desingularization carried out through the displacement of collocation nodes outside of the domain, as in [106].

Our FMM implementation is fully adapted to π -BEM characteristics, and automatically considers possible singular quadrature schemes among the close range source-target interactions. This choice guarantees that the only error we are introducing is due to truncation of the harmonic series in the FMM which is bounded by well known *a priori* estimates as shown in [46].

The FMM uses a *divide et impera* strategy combined with multipole and local (Taylor-like) expansions to evaluate the long range interactions. Once these ones have been approximated with the procedure sketched in Figure 3.5, we evaluate the short range interactions directly using the exact pairwise formulation.

Referring to Figure 3.5 we can identify 5 main phases in which the FMM can be subdivided.

1. *Hierarchical Tree Generation* required to partition the domain in an *octree* blocking.
2. *Direct Short Range Interactions* to compute the actions of quadrature source points on nearby collocation nodes.
3. *Ascending Phase* to compute the Multipole expansions in the childless blocks and translate them into higher level blocks (black dashed and red dotted lines in Figure 3.5).
4. *Descending Phase* to convert the Multipole expansions to the Local expansion of well separated blocks, to transfer them into lower level blocks and to evaluate them for each childless block on any collocation node (blue dashed, green dotted lines and magenta dashed lines in Figure 3.5).
5. *Sum of the Contribution* to combine short range and long range interactions and retrieve the complete matrix vector products.

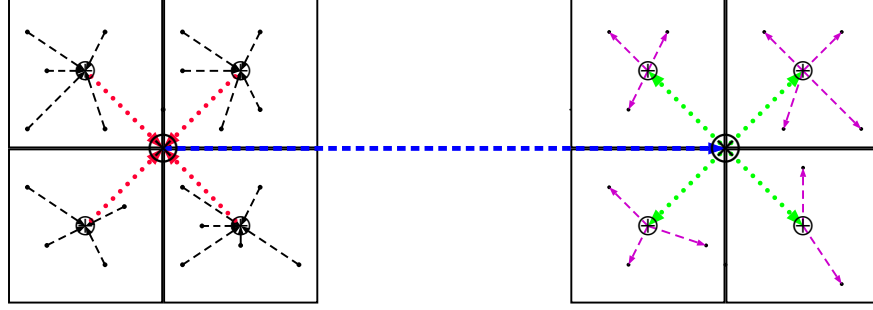


Figure 3.5: The sketch of long range interaction computations in the FMM framework. On the left we have $M = 20$ sources, on the right $N = 12$ evaluation nodes. On each childless block a Multipole expansion is generated including the effects of local sources (black dashed lines). The Multipole expansions are then translated to the parent blocks (red dotted lines). The blue dashed line represents the conversion of the multipole expansion to the Local expansion of the well separated block. The far away interactions summarized by such Local expansion are then transferred to children blocks through Local expansion translations (green dotted lines). Finally Local expansions of each childless block are evaluated in correspondence with all target nodes (dashed magenta lines).

3.2.2 FMM validation

We validate our BEM-FMA implementation, showing that we obtain the required accuracy and computational cost on all problems previously investigated with the direct BEM formulation. A first assessment is presented in Figure 3.6 which shows a comparison of the computational cost between standard BEM (blue lines), and FMM-BEM (red lines). The left plot refers to the CPU time required for a single matrix vector multiplication as a function of the number of unknowns. The right plot compares the CPU time required to set up the problem solution for both solvers as a function of the number of unknown. In the standard solver the setup time corresponds to the time required to assemble the BEM matrices. In the BEM-FMM solver, the setup time includes the *octree* partition of the domain, the computation of the close range interaction matrices, and the allocation of the data structure for the Multipole expansions. As reference we plot with dashed and dash-dotted lines the linear and the quadratic computational costs.

Both plots confirm that the FMA allows for a linear computational cost for both the problem set up and matrix vector products. It is worth pointing out that the quadratic computational cost associated with the standard BEM solver leads to more competitive performance when a small number of unknowns is considered. The breakeven point is roughly located at 10^4 unknowns.

The FMM algorithm introduces, in the computation of a single matrix vector product, an error associated with the truncation of the harmonic series expansions. A fundamental feature of the original algorithm proposed by Greengard and Rokhlin, see [46], is that it is possible to derive a global bound for such error, namely

$$e_{FMM} \leq C \left| \frac{r}{z} \right|^{p+1} \leq \left(\frac{1}{2} \right)^{p+1}. \quad (3.1)$$

In equation (3.1), r represents the radius of the sphere used to generate the Multipole

Chapter 3. Parallel implementation of a Boundary Element Method and its acceleration using a Fast Multipole Method

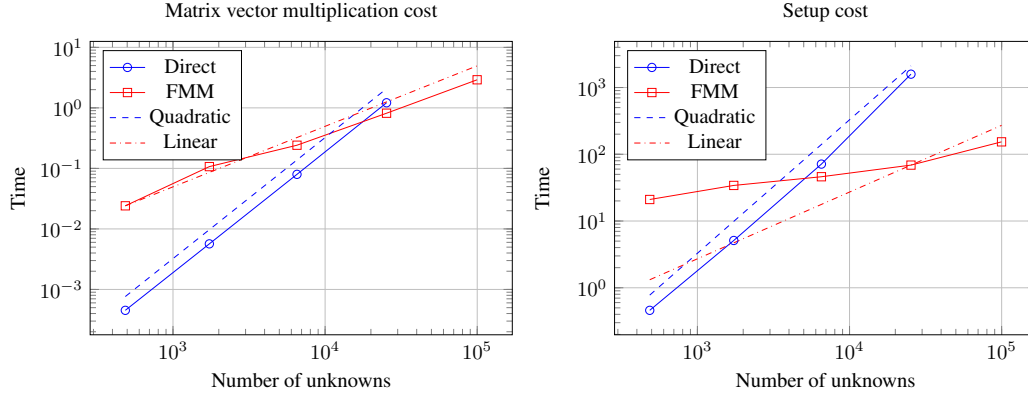


Figure 3.6: Computational Cost comparison between BEM and BEM-FMM. The figure on the left reports a comparison of a single matrix vector multiplication, the one on the right reports a comparison of the setting time needed by the method. In the direct solver the setup time corresponds to the assembling time for the BEM matrices. In the BEM-FMM the setup corresponds to the octree generation, the computation of the close range interaction matrices, and the allocation of the data structures required by the Multipole expansions. The blue circled curve represents the direct timing, the red squared one the accelerated method analysis. dashed and dash-dotted lines depicts linear and quadratic computational costs.

expansion, z is the minimum distance between such sphere and the evaluation node, while p is the truncation order of the Multipole expansion. In FMM algorithm the expansion is only used for sources and nodes lying in well separated boxes, for which $\frac{r}{z} \leq \frac{1}{2}$. We verify that the present FMM method is able to recover the expected convergence to the standard BEM solution, by considering the pyramid test case presented in Section 1.3.1 with 486 degrees of freedom and we let the terms of the harmonic expansions vary from 2 to 10. In Figure 3.7 we report the convergence rate of the FMM to the direct matrix vector product evaluation (left plot) and to the standard BEM solution (right plot). The blue lines with circles represents the L_2 error as a function the FMM truncation order. The green lines with squares indicates the L_∞ norm of the error as a function the FMM truncation order. The red lines with triangles represents the reference exponential error bound reported in (3.1).

The matrix vector accuracy plot confirms that the BEM-FMM matrix vector product converges with the expected behavior to the direct matrix vector product evaluation. The right plot ensures that such accuracy is not lost through the several matrix vector products needed for the convergence of the iterative GMRES solver.

To validate our BEM-FMM implementation when local refinement strategies and high order methods are used, on geometries with sharp edges, we repeat the test case presented in Table 1.1, which refers to the mixed boundary condition problem on the pyramid geometry using high order elements with adaptive local refinement. The results are presented in Table 3.2 which shows identical results with respect to those reported in the corresponding direct BEM Table 1.1, except for the CPU times.

3.2.3 FMM hybrid parallelisation

In recent years many different parallelization strategies have been developed for the FMM. A pure MPI parallelization would imply the communication of all the complex

3.2. Accelerated Boundary Element Method

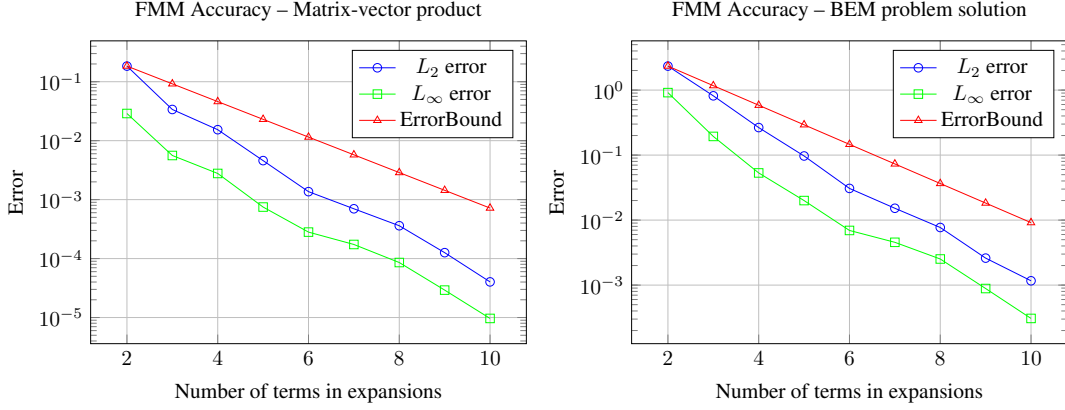


Figure 3.7: Convergence Analysis for the Multiple expansion using 486 unknowns on the pyramid test case. On the left we report the error of a single matrix vector multiplication, on the right we plot the error on the overall solution. We plot in blue with circles the L_2 error of the current FMM implementation and in green with squares its L_∞ norm, as reference we depict in red with triangles the expected exponential decay.

Table 3.2: Comparison between different continuous Finite Elements using local refinement strategy in the BEM-FMM framework considering 6 terms in the expansions. We consider a reference error of 5×10^{-5} and we compare both the number of unknowns and the required computational time.

Element	$L_\infty \phi$ Abs Error	Cells	Unknowns	Time (sec)
Q_1	4.53×10^{-5}	2168	2526	60.4
Q_2	3.35×10^{-5}	351	1672	14.41
Q_3	2.02×10^{-5}	174	1857	15.32

data structures regarding Multipole expansions at each level of the hierarchical tree. In particular the need of a parallelization strategy that couples different paradigms has been spotted in recent years in the framework of parallel multipole methods [121]. This issues has been successfully addressed by the outstanding work by Yokota *et al.* [121], or by Malhotra *et al.* [71]. In [121] different sets of processors have been set up to properly split the workload among the tree in the ascending phase, taking care in a scalable way of the complex communication pattern. In many FMM application, e.g., [65], the first key-step is the computation of the so called *Locally Essential Tree* that drives the overall MPI communication. Our implementation choice, described in Section 3.1, exploits the robust and automatic domain decomposition described in Section 3.1.2. This choice allows for a relatively straightforward interface between the BEM solver and the FMA, however, it has the drawback that we can't guarantee the construction of such hierarchical subdivision at the tree level. We profile the execution of the accelerated BEM-FMA, as we did for the standard BEM, in Section 3.1, using the BEM-FMA solver on a single CPU. In this first computation we consider a single node per leaf in the octree. We execute a computations with 6144 cells and 6534 degrees of freedom using bilinear Q_1 FiniteElement, and we report the overall time, as well as the timings of the four main part of the algorithm: tree generation, computation of near range interactions (direct interaction), ascending phase and descending phase.

Chapter 3. Parallel implementation of a Boundary Element Method and its acceleration using a Fast Multipole Method

Table 3.3: Profiling of BEM-FMA solver on a single CPU.

Method	Time (sec)
Tree Generation	1.798
Direct Interaction	5.723
Ascending Phase Time	0.533
Descending Phase time	9.258
Total Time	19.92

From Table 3.3 we see that the most demanding part of the algorithm is the computation of the direct contributions, and of the descending phase. Due to the lack of a locally essential tree, a purely MPI based parallelization would require too much communication, and too much replicated data structures among nodes. On the other hand, by carefully combining shared and distributed parallelization, we mitigate the lack of a tree structure in our FMM algorithm.

We apply multithreaded parallelization extensively to every part of the code, and we restrict the distributed memory parallelization to the most demanding phases of the algorithm, by enforcing that the number of threads and the number of MPI processes is kept well balanced at all times. In π -BEM we don't address yet the work balance between different processor in the ascending phase. This choice is driven by the fact that we are interested in a relatively small number of unknowns (with respect to the exascale FMM presented in [121]) of the order of $O(10^5)$. Instead, both the direct interactions and the descending phase are parallelized combining multithreaded and multi-processors techniques. Given the relatively small size of the problems, compared to exascale applications, shared memory parallelism is for the moment sufficient for the remaining parts of the FMM. Improvements in this direction are under study.

The ascending phase is replicated on each MPI processor. Once the ascending phase is performed, each processor performs the descending operation on a given block only if it contains a degrees of freedom owned by the processor itself. For short range interactions we use the same techniques shown in Section 3.1 to parallelize using both distributed and shared memory parallelism. A purely shared memory paradigm is applied to all the other part of the FMM. We use Intel Threading Building Blocks for local shared parallelization [99]. In the present work we follow the techniques presented in [115] to guarantee the absence of racing conditions when using shared parallelization.

The FMM can be effectively tuned for a parallel computation, see for instance [45]. One of the most important parameters in the FMM is the number B of evaluation points we allow per leaf of the tree. In particular if we consider a serial run on the pyramid test case using 1536 degrees of freedom we get very different result varying the parameter B . If we consider $B = 1$ we get a computational time of 130 seconds, while if we consider $B = 50$ we retrieve the solution in only 17 seconds. We want to get some insights on the better tuning of the parameter B to ease the overall computational time of the BEM-FMA algorithm. Greengard and Gropp derived a simplified analysis of the computational costs to recover an optimal size for the block. We repeat such simplified analysis for the presented FMA in the hybrid shared (TBB) distributed (MPI) memory framework, namely we analyze the computational costs of each section of the algorithm and we retrieve an insight for the best setting of the parameter B .

We define the following quantities:

$$\begin{aligned} N &= \text{number of unknowns,} \\ B &= \text{number of unknowns in a leaf,} \\ p &= \text{number of MPI processors,} \\ t &= \text{number of TBB threads.} \end{aligned} \tag{3.2}$$

The time needed by the FMM can be sketched as

$$T_{FMM} = T_{ascending} + T_{descending} + T_{direct} + T_{comm}, \tag{3.3}$$

where $T_{ascending}$ represents the time needed by the ascending phase, $T_{descending}$ is the time needed by the descending phase, T_{direct} represents the time needed by the short range interactions, while T_{comm} is the time required by the parallel communications or synchronisations. If we expand all the terms of equation (3.3) we obtain,

$$T_{FMM} = K_1 \frac{N}{t} + K_2 \log_8 \left(\frac{N}{B} \right) \frac{N}{Bt} + K_3 \log_8 \left(\frac{N}{B} \right) \frac{N}{Btp} + K_4 \frac{N}{tp} + K_5 \frac{NB}{pt} + e(B, p, t), \tag{3.4}$$

where $e(B, p, t)$ is the function representing the communication costs and is in general a function of B, p, t and depends heavily on the architecture we are exploiting for the parallel computation, K_1, K_2, K_3, K_4, K_5 are fitting constants that depend on the precision requested to the FMA and on the characteristic of the underlying computational architecture considered. We follow [45] and determine them experimentally. We look for a stationary point of the total cost, by computing the partial derivative of all the timings with respect to B of (3.4) and we search its root. We get the following estimate, assuming that $\partial e / \partial B = 0$,

$$B_{opt} = \sqrt{\frac{(K_2 + K_3/p) \log_8(N)}{K_5/p}}. \tag{3.5}$$

The optimum depends on the number of MPI processors we uses, this is due to the fact that our hybrid parallelization exploits differently distributed and shared memory paradigms. The optimization represented in (3.5) is a simplification of the real optimum since we are neglecting the variations of the communication costs, however it can provide some insight for the setting of a more efficient parallelization

To determine the fitting constants K_1, K_2, K_3, K_4, K_5 We use a single MPI processor and we exploit the maximum number of threads on two 10 cores (E5) Intel Xeon E5-2680 v2. To minimize the penalization due to the lack of a well parallelized ascending phase, we use 1 MPI process per node, with 20 independent parallel threads per node. We test block sizes from 20 to 140 nodes per block. We retrieve, by means of a least square fitting of experimental data, the constants K_1, K_2, K_3, K_4, K_5 . In Figure 3.8 we plot the overall approximation, in blue with circles we plot the expected behavior coming from (3.4) while the green squared line shows the real experimental behavior we have measured. From Figure 3.8 we see quite a good agreement for small block sizes, then we see a clear mismatch between the simplified theoretical analysis and the experimental results. We believe that the disagreement is due to complex effect of the present hybrid parallelization, for example the increased size of the blocks could

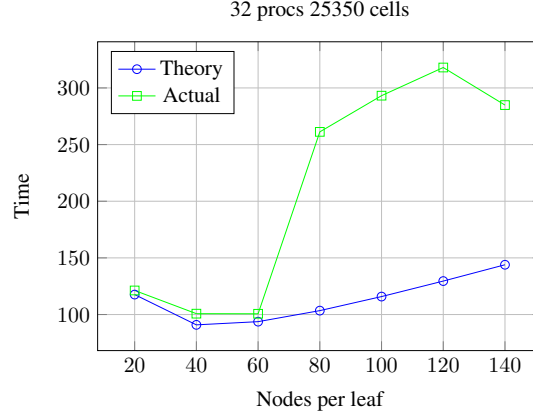


Figure 3.8: Analysis of the time needed by the overall FMA. We consider 1 MPI processors and up to 10 threads per processor. We let the number of evaluation nodes per block B vary from 20 to 140. In blue with circles we report our theoretical result using the preconditioner timings, in green with squares we can see the experimental results.

easily lead to an increase of the number of cache misses in our code, this may induce a non negligible performance loss. To prove our hypothesis we have carried out a simple profiling of the performance of the algorithm. In Table 3.4 we report some meaningful events of the algorithm, namely cache-misses and branch-misses.

Table 3.4: Profiling of the time for the nodes per leaf analysis.

Event	60 node per leaf	80 node per leaf
cache-misses	1077312088	2588937415
branch-misses	7565628895	8433064720
Direct time	25.58	110.9

From Table 3.4 we see that passing from 60 to 80 nodes per leaf causes an increase in the number of cache misses of more than 100%. The increase number of the misses deeply worsen the overall performance of the Task Scheduler upon which the shared memory parallelisation is based, see [99, 115]. With our theory we get

$$B_{opt} = \sqrt{\frac{(K_2 + K_3/p) \log_8(N)}{K_5/p}} = 52.5440911254, \quad (3.6)$$

which agrees with experimental results that show the minimum for $40 < B_{opt} < 60$. In Section 3.2.4 we will therefore consider $B = 60$. We highlight that B_{opt} depends on the number of MPI processor we use. However, since our FMA is mostly shared memory parallelised we are not interested in requiring many MPI processors, thus we believe $B = 60$ to be a good choice for the scaling analysis up to 2 computational nodes composed by two Intel Xeon E5-2680 v2 each.

3.2.4 FMM: strong scaling

We study the Strong Scaling of our BEM-FMA algorithm with hybrid TBB-MPI parallelization up to 2 nodes, each node is composed by two 10 cores (E5) Intel Xeon

E5-2680 v2. We consider a problem with 98306 degrees of freedom. To better highlight the performance of our implementation we plot two different scaling analysis in Figure 3.9. On the left we depict the strong scaling analysis of the overall algorithm and on the right we draw the scalability concerning a single matrix vector multiplication. In the left graph we plot in blue with dots the overall scalability, in green with squares the scalability for the complete set of matrix vector multiplications, in magenta with triangles the scalability for the setting time of FMM, in cyan with stars the scalability of the preconditioner building method. On the right we represent in blue with circles the overall scalability, in green with squares the scalability descending phase of matrix vector multiplications, in magenta with stars the scalability of the setting time and in black with triangles the scalability of the ascending phase of the algorithm. The ideal scalability is represented using red dots. The analysis and comparison of the two plots of Figure 3.9 allows for a better understanding of the bottlenecks concerning the overall BEM-FMA and the present FMM implementation.

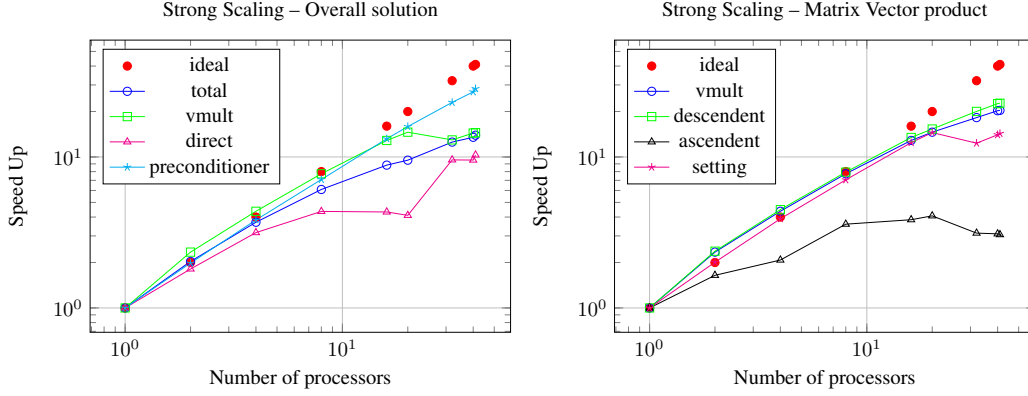


Figure 3.9: Strong Scalability test using 98306 degrees of freedom. We test up to 40 processors. On the left we plot the scalability of the overall algorithm and on the right the scalability of a single matrix vector multiplication. We report the scaling considering the worst timing on all the used processors. For what concerns the plot on the left we plot in blue with circles the overall scalability, in green with squares the scalability for the complete set of matrix vector multiplications, in magenta with triangles the scalability for the setting time of FMM, in cyan with stars the scalability of the preconditioner building method. In the right figure we plot in blue with circles the overall scalability, in green with squares the scalability descending phase of matrix vector multiplications, in magenta with stars the scalability of the setting time and in black with triangles the scalability of the ascending phase of the algorithm. In both plots we plot as reference the ideal scalability using red dots.

We get an overall scalability of 13.93, with a scalability issue for the short range interactions, and the matrix vector products. Part of the performance loss is induced by the augmented number of iterations needed by the iterative solver when MPI is involved, as suggested by the right plot of Figure 3.9. That plot shows how our preconditioner is not optimal w.r.t. number of MPI processes, obtaining a speedup of 20.36, which is almost double with respect to the the overall scalability, when using a single MPI process.

We are currently studying ways to improve the efficiency and the performance of the developed BEM-FMM algorithm. We have implemented the algorithm to ease the main problematics of the standard BEM: the quadratic cost and the memory requirements. While a linear computational cost is guaranteed by a proper implementation of

the FMM, see Section 3.2.2, we have chosen to compute at run-time all the operations needed by the long range interaction approximations required by the FMM. This choice minimises the memory requirement but it may induce a severe overhead for every matrix vector multiplication. In many FMM implementation, for instance in [20], most of the matrices representing operations as Multipole to Multipole or Local to Local translations are precomputed. This strategy increases the memory requirement and the setup time for the method but eases the matrix vector product CPU timings, especially in a parallel environment, since matrix vector product operations can be highly optimized by existing HPC libraries. We are currently studying the data structures and algorithm modifications to investigate such an implementation strategy.

However, we believe that the most consistent improvement may come from a more efficient preconditioner. Classical studies, for instance [87, 113], proved the possibility of exploiting Multi Grid techniques coupled with BEM. Such preconditioners may significantly reduce the number of iterations required for the iterative solver to converge, and would ease also the aforementioned overhead due to the run-time computation of the FMM data structures. The implementation of a better precondition strategy for the considered mixed Dirichlet-Neumann problems is undergone at the moment.

Parallelization of type 3 Non Uniform FFT and its application in fast numerical convolution

Fast Multipole Method is not the only a possible accelerator for the Laplace BEM. In recent years many different methods have been developed exploiting the Fast Fourier Transform of the data. In particular in [3] Alouges and Aussal presented a new algorithm called Sparse Cardinal Sine Decomposition (SCSD) to accelerate the standard Laplace BEM. The results are very promising and comparable to the FMM. SCSD is based on Fourier transforms between arbitrary points in space and frequency domains. In this Chapter we present a parallel implementation of the particular FFT required in SCSD algorithms, and a parallel version of SCSD itself. SCSD is based on Fourier transforms between arbitrarily-spaced grids.

The standard Fast Fourier algorithm relies on a distribution of the points on a regular equi-spaced grid. However, many applications benefit from an accurate and reliable Fourier transform between N arbitrarily spaced points. The computational cost of the standard discrete transform increases quadratically, quickly becoming unbearable even on modern computational platforms.

A solution to this problem is the Non Uniform Fast Fourier Transform (NUFFT) developed by Dutt and Rokhlin [31]. The authors provide a deep theoretical analysis to approximate the Fourier transform using the classical Fast Fourier Transform algorithm. Another description of such algorithm has been addressed by Leslie Greengard and June-Yub Lee in [47]. In the literature there are 3 different type of NUFFT: type 1 operating between arbitrary points in space and a regular grid in frequency, type 2 relating a regular grid in space and arbitrary points in frequency, and type 3 dealing with arbitrary points in both the spaces.

The key idea of NUFFT is to transfer the non equispaced data on a uniform grid to be used with standard FFT algorithms. In [47] the authors use fast convolutions with

a Gaussian function to create a uniform grid where standard Fast Fourier Transform algorithms can be used. This is a key step to retrieve the solution in an affordable time, since Gaussian convolution can be efficiently computed by means of Fast Gaussian Gridding, reducing the overall computational time. Another possible solution is the usage of Kaiser-Bessel window or the min-max interpolator. In recent years the latter has been efficiently implemented in the library NFFT, see [92]. In [12] the authors propose a parallelization of type 1 NUFFT based on the P3DFFT, see [89] for further details. NUFFT of type 3, developed in [47], has been successfully applied in fast numerical convolutions [3], and has applications in many fields of engineering. Several parallelizations have been performed for such an algorithm, in [61, 108] highly scalable optimization of the library on modern multicore systems are proposed, in [105] the authors use the hardware to accelerate the NUFFT, while in [83] the multicore architecture of GPUs is used to tune and optimize NUFFT.

While the previous implementations have proved to be effective they are extremely hardware specific and optimized. Moreover, their modification is far from trivial from the user's perspective. We propose an OpenSOURCE, released under GPL license, flexible parallelization strategy of the type 3 NUFFT algorithm based on existing High Performance Computing libraries, we call such a library BlackNUFFT [39]. We use Intel Threading Building Blocks (TBB) for shared memory parallelism [99, 115], together with the standard Message Passing Interface (MPI) for distributed parallelism. High modularity is a key aspect of our implementation, allowing the user to plug-in new algorithms for each aspect of NUFFT. We use a black box approach for FFT on the fine grid making it possible to interchange backend libraries for the FFT algorithm. We provide an implementation that uses by default FFTW [33] as backend FFT. This choice is mainly due to the extensive documentation and high reliability of such a library. We compare the presented BlackNUFFT with the library developed by Lee and Greengard which is freely available under GPL license, see [67]. We use existing implementation of distributed vector provided by existing OpenSOURCE libraries. In particular we refer to the implementation of the `deal.II` library, see [9–11], which provides both high performance on modern architectures and the possibility of easily switching between 32 and 64 bits indexing. This capability is a keystone to reach higher computational complexity. We present a performance analysis considering multithread and multiprocessor environments for the current implementation of the library. BlackNUFFT is available under GPL license on GitHub [39].

Finally we present a possible application of BlackNUFFT library to Sparse Cardinal Sine Decomposition, applied to Boundary Element Method in [3]. Such algorithm shows very promising results and the present Chapter presents a preliminary study of its parallelization.

In Section 4.1 we group some considerations on theoretical aspects of NUFFT [31, 47] and we describe the actual structure of the algorithm. In Section 4.2 we analyze the efficiency of shared memory parallelism alone. In Section 4.3 we combine shared and distributed memory paradigms to reach higher level of scalability. In Section 4.5 we present the SCSD algorithm and its parallelization using BlackNUFFT.

4.1 NUFFT type 3

4.1.1 Mathematical background

We follow [31, 47] to introduce the main aspects of NUFFT of type 3. Given a set of N complex values in the three dimensional space $f(\mathbf{x})$ we define its Discrete Fourier Transform to a set of N points in the frequency space as

$$F(\mathbf{k}) = \frac{1}{N} \sum_{i=0}^{N-1} f(\mathbf{x}) e^{-i\mathbf{x} \cdot \mathbf{k}}, \quad (4.1)$$

with $\mathbf{k} = (k_1, k_2, k_3)$ and $\mathbf{x} = (x_1, x_2, x_3)$. We also have that $-N/2 \leq k_1, k_2, k_3 < N/2 - 1$. We can define also the inverse DFT as

$$f(\mathbf{x}) = \sum_{i=-N/2}^{N/2-1} F(\mathbf{k}) e^{i\mathbf{x} \cdot \mathbf{k}}. \quad (4.2)$$

DFTs defined in (4.1) and (4.2) are characterised by a computational cost $O(N^2)$. NUFFT algorithms are based on an interpolation scheme of the arbitrary points in real and frequency space to a regular grid on which we can apply the standard FFT algorithm. In the present work we achieve this task by the so-called Gaussian Gridding algorithm to perform this interpolation, see [47]. An alternative gridding techniques is the min-max interpolator, which has proved to be very efficient especially when the dimensionality of the operation increases, see [92]. In the case of Gaussian Gridding we exploit the following one dimensional estimate to approximate the exponential e^{ikx} function on the regular grid

$$\left| e^{icx} - e^{bx^2} \sum_{l=c-q/2}^{c+q/2} \frac{1}{2\sqrt{b\pi}} e^{-(c-l)^2/4b} e^{ilx} \right| < e^{bx^2} e^{-b\pi^2} (4b + q) = \epsilon, \quad (4.3)$$

where c represents the nearest point to k on the regular grid, q is usually referred to as spreading and b is a scaling constant. Estimate (4.3) trivially extends to the three dimensional case. We need to set variables b, q in order to assure that the estimate fulfils the accuracy ϵ for any points in our arbitrary distribution. The fact that we can a priori set the accuracy of the computation is a key point of the entire algorithm, and in general of all efficient fast accelerated method, [46]. The summation in (4.3) can be seen as a convolution like $\delta(x - \bar{x}) * g_{\bar{x}}(x)$ where $g_{\bar{x}}(x)$ represents a Gaussian like function centered in \bar{x} . Such a convolution can be efficiently performed using the Fast Gaussian Gridding (FGG) depicted in [47]. Once we have obtained a regular grid we apply a FFT to obtain a transformed regular array, and apply again an interpolation to retrieve the result on an arbitrary output grid. We do this with another FGG using again (4.3) to have an accurate computation.

4.1.2 Algorithm key-steps

We have divided the algorithm in 4 key-steps:

1. Computation of the bounding box for the arbitrary grids, and set up of the griddings in order to retrieve the requested accuracy. This operation is not demanding from a computational point of view since it only sets up the spreading constants for the griddings and the dimension of the fine grid array.
2. Transfer of the original data on the uniform grid. Since we have chosen Fast Gaussian Gridding we have:
 - Fast Gaussian Gridding of the original data to the regular fine grid.
 - Scaling to correct the Gaussian Gridding to be performed on the output array.
3. FFT on the regular grid: basically a 3d FFT pruning the border values (thus saving computational time) that represent convolution errors, a data shift keeps the low frequencies at the center of the spectrum
4. Transfer of the transformed data from the uniform grid to the output points. Since we have chosen Fast Gaussian Gridding we have:
 - Fast Gaussian Gridding from the fine grid to the output data.
 - Scaling on the output array to correct errors introduced by the first gridding.

The algorithm uses two griddings between the input data and the uniform grid and between the uniform grid and the output data. As preliminary choice we use Fast Gaussian Gridding to perform both. These operation can be highly optimised by means of a Fast Convolution algorithm, making its implementation not trivial even on a single core. We designed the parallelization to work with any parallel implementation of the FFT. We only need the chosen library to provide a suitable 3D subdivision of the fine grid array. In particular we use the three dimensional implementation of FFTW. Even the single griddings can be easily interchanged, we just need to guarantee the creation of the fine grid array. In Section 4.4 we describe the procedure to introduce such new features inside the presented library. Figure 4.1 depicts the structure of the presented library.

4.2 Shared memory parallelism

All modern CPUs support multicore shared memory parallelism. We use Intel Threading Building Block [99] to exploit this possibility and achieve higher efficiency. This tool has been successfully adopted in many high performing library, as ASPECT [63], or the `deal.II` library [6]. Moreover it introduces the use of the TaskScheduler concept that allows for higher level of optimization in our library. We follow the shared memory parallelization strategies introduced in [115] and we apply it all over the NUFFT key-steps.

4.2.1 Implementation

We profiled our serial library on the following test case scenario, a single forward NUFFT from an array of $2^{21} = 2097152$ randomly distributed points to another array of $2^{21} = 2097152$ randomly distributed points, we consider an accuracy of $\epsilon = 1 \times 10^{-5}$. It is well known that the spreading constant for the determination of the fine uniform

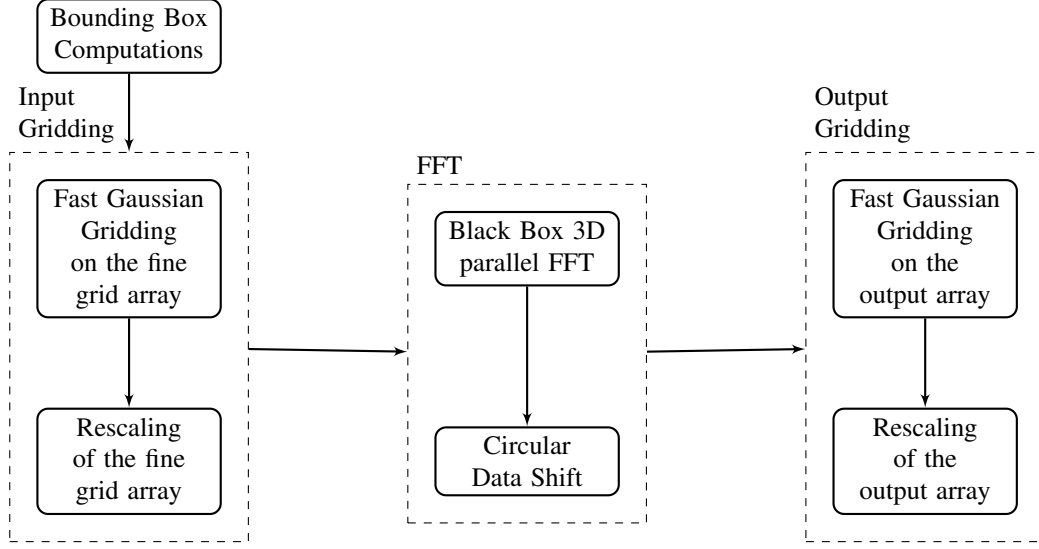


Figure 4.1: Flow diagram depicting the structure of the library. It can be roughly subdivided in three main parts, and we stress that each of them can be substitute and customised by the user.

Table 4.1: Profiling of a serial run of BlackNUFFT application.

Function	Time (sec)	%
FGG on Inputs	60.1	45
Scaling on Inputs	4.25	3.2
FFTW 3D	18.7	14
Circular Shifting	3.13	2.4
FGG on Outputs	39.8	30
Scaling on Outputs	2.14	1.6

grid are influenced both by the input and the output array, thus for the sake of clarity we consider the following relationship between the maximum frequency K_{max} and the maximum spreading R_{max} in the original space,

$$\frac{R_{max}K_{max}}{\pi} = 100. \quad (4.4)$$

This is considered to be a realistic range for a NUFFT [3]. We get a fine grid array of 373248000 points, requiring approximately 6 GB of memory. The profiling is reported in Table 4.1. The most demanding functions are the two griddings (64.35 and 41.94 seconds respectively) together with the three dimensional FFT (23.7 seconds).

Input gridding The most demanding part of the algorithm is the Fast Gaussian Gridding. We focused on this particular algorithm for the extensive literature and high accuracy it can guarantee [47]. However, we highlight that this is a preliminary study and the study of different efficient griddings, especially the min-max algorithm [92], is undergoing at the moment. We compute the first gaussian gridding from the input array to the finer grid. Basically this is a convolution through a Gaussian kernel. We

perform it following the Fast Gaussian Algorithm developed by Greengard, see [47]. This part of the algorithm presents several race conditions in the writing of the fine grid array, therefore we expect some parallelization issues. A subdivision of the fine grid array reduces the synchronization requirements. The accuracy ϵ we prescribed settles the span of the Gaussian kernel to be used in the FG, ϵ defines the number of point q that a generic input point can influence through the gridding. We subdivide, in any direction we prefer, the fine array in sections of span $2q$, and we consequently create index sets for the input array that contain the elements that have the nearest fine point in each subdivision. In this way we compute alternate odd-even subdivisions without any race conditions between different even-even or odd-odd subsets. We manage the scheduling of the different threads using the standard parallel loop described in [99]. We split the array along its second dimension, leaving the first one to set up the MPI parallelization described in Section 4.3. A summary of the required steps for the shared memory parallelization follows.

- Subdivision of the fine array along one single dimension. We sketch the subdivision in Figure 4.2. If we consider a gridding radius q we can split the fine grid array using $2q$ to identify elements that will not have any racing conditions. In Figure 4.2 we see that all the region with the same colour can be written at the same time.
- Subdivision of the input array between different subdivisions.
- Gridding of all the even, depicted yellow in Figure 4.2, subdivisions using TBB in each subset.
- Gridding of all the odd, shown in blue in Figure 4.2, subdivisions using TBB in each subset.

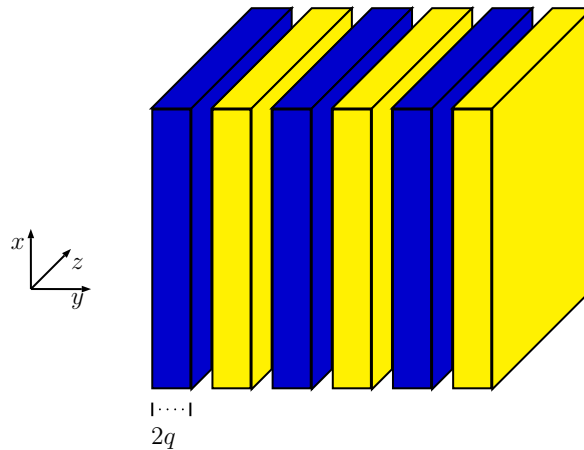


Figure 4.2: *Subdivision for the shared memory parallelization.*

We don't expect an optimal scalability for two main issues: the setting of the parallelization introduces a slight overhead, and since we consider an oversampled FFT there are empty sets at the boundaries. This situation unbalances the workload between different threads, introducing suboptimality.

To complete the gridding of the inputs we need a scaling of the fine grid array. With this operation we satisfy equation (4.3), that is, we need a scaling to correct the gridding on the input array. This is a pointwise multiplication by a factor e^{bx^2} , not presenting any parallelization issue.

Compute 3D FFT and data shift We perform a FFT on the fine grid array exploiting the backend HPC library of our choice. We enable the multithreading in the external FFT library, so we simply need to set the number of threads to be used. In FFT it is common to shift the data in order to have the lowest values at the center of the grid. We apply a standard circular shift, which is a local embarrassingly parallel operation. We just need to apply the transformation depicted in (4.5) to the transformed three dimensional matrix $F(i, j, k)$, namely

$$F(i, j, k)_{shift} = -1^{i+j+k} F(i, j, k) \quad i, j, k = 0 \dots N_1 - 1, N_2 - 1, N_3 - 1, \quad (4.5)$$

where N_1, N_2, N_3 represent the dimension of the fine grid. We stress that shifting the input fine matrix would be unbearable especially in distributed memory as in Section 4.3, where we would need to communicate GigaBytes of data.

Output gridding This operation can be seen as the adjoint of the first input gridding. We need to recover the data on the output array starting from the fine grid representation, and we use TBB to exploit the multicore parallelism. Since we only have concurrency in reading we expect almost an optimal behavior for this step and we don't need any particular subdivision strategy for the parallelization.

We scale the data to correct the Gaussian gridding we have performed on the output array. This is still a pointwise operation so we use a simple shared memory implementation with no race condition handling.

4.2.2 Strong scaling analysis up to 16 threads

For the analysis of the strong scalability of our pure TBB parallelization we run the computations on a single node, with a Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz processor with 16 hyperthreaded cores. We consider the same NUFFT setting we aforementioned in Section 4.2.1, and present the scalability results in the left plot of Figure 4.3. On the right we plot the relative importance, in terms of computational time, of each function composing the NUFFT algorithm.

We see that the fast gridding on the output array, the two scaling operations, and the shift after the FFT show almost an optimal behaviour. This is expected since these are local operations and almost no write racing condition occurs. However, we see that the parallelization of the fast gridding from the input array is suboptimal. This is due to the setting up time of the parallelization strategy which we have sketched in Figure 4.2. To reduce the number of racing conditions we need to carefully subdivide the input array in the different subsets, and this operation introduces a slight overhead. Moreover, since we are considering an oversampling, there will be some empty sets at the boundaries of the fine grid array. These empty subdivisions, together with the setting up time, unbalance the overall workload between different threads inducing the suboptimality. We see that the three dimensional FFT, which has a multithreaded parallelism enabled, behaves almost optimally. Since multithreaded performance relies

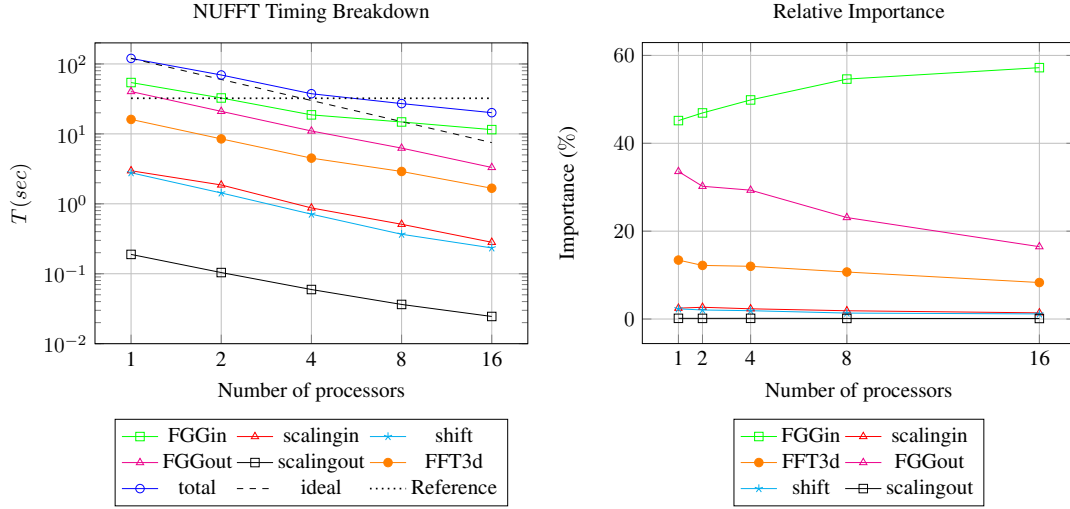


Figure 4.3: Performance analysis of the parallelization using shared memory paradigms. We consider a single forward operation on 2097152 arbitrary points. On the left we plot the timings of all the functions. The orange line with dots represents the 3D FFT transformation, the blue circled line is the overall timing, the green squared one depicts the first FGG, the red triangled plot is the scaling before FFT. The starred cyan line is the shift of the data, the magenta triangled line depicts the second FGG while the black squared line is the scaling after the FFT. The black dashed line represents the optimal linear scalability. The black dotted line represents the timing of the chosen reference NUFFT library [67]. On the right we plot the relative importances of the different functions. The orange line with dots represents the 3D FFT transformation, the green squared one depicts the first FGG, the red triangled plot is the scaling before FFT. The starred cyan line is the shift of the data, the magenta triangled line depicts the second FGG while the black squared line is the scaling after the FFT. We let the number of threads vary from 1 to 16.

on appropriate choices for the number of tasks, the granularity, and probability of task stealing, we are currently testing different choices of the granularity in our parallelised for loops. The overall time required by BlackNUFFT is greater than the time required by the reference serial FORTRAN library only if we require less than 8 threads. Fortran has strict aliasing semantics compared to C++ and has been aggressively tuned for numerical performance, for this reason we expect some performance advantage for the reference library if we don't fully exploit our parallelization.

To better understand the actual importance of each function we draw in the right plot of Figure 4.3 their relative impact from a computational point of view. We note that the TBB parallelization drastically reduces the computational time of all functions. The gridding on the input array is the real bottleneck of the algorithm since it shows the least optimal behavior. To reach higher computational complexities and reduce even further the computational time, we implement a parallelization strategy based on hybrid shared and distributed memory environment that reduces the overall computational time without losing the benefits of the shared memory parallelization introduced so far. A hybrid parallelization, combining MPI and TBB, effectively reduces the overall computational time while maintaining a good overall scalability.

4.3 Distributed memory parallelism

We use the standard Message Passing Interface between different processors to obtain distributed memory parallelism, and to overcome the two main bottlenecks of the multicore parallelization: the input gridding (through FGG) and the three dimensional execution of the FFT on the fine grid. We start by presenting the coding paradigm we have followed and we explain in details each function of NUFFT, and finally we analyze the scaling performance of our new implementation.

4.3.1 Index sets creation

The 3D FFT is the core of the algorithm and we let the backend FFT library determine the unknown splitting by different MPI processors. We adopt the same unknown splitting over the NUFFT process. We focused our attention on the use of the three dimensional parallel implementation of FFTW and we computed the unknown splitting required by the FFTW MPI routine. This is a simple one dimensional domain decomposition along the coarsest dimension (see Figure 4.4), and we let the library itself determine the decomposition among the coarsest coordinate of our domain. Subsequently such decomposition is used as a starting point of the overall parallelization. We determine the workbalance among different processors according to the backend FFT library, and assign the input and output array entries to each processor depending on where their nearest points are placed on the overall fine grid.

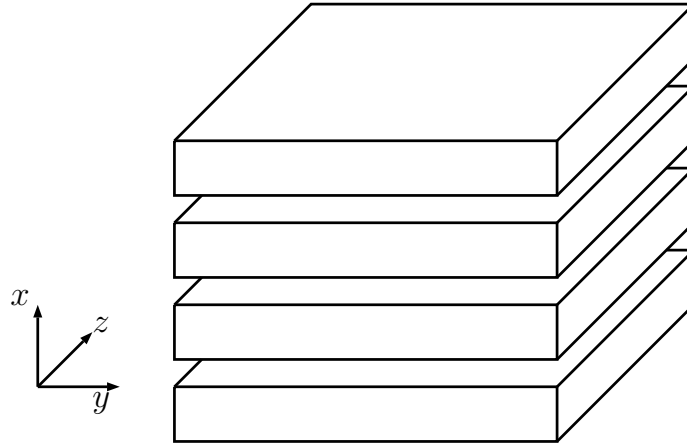


Figure 4.4: Subdivision for the distributed memory parallelization.

The Gaussian gridding we use has a span q determined by the requested tolerance, and we use this spread to determine the ghost levels of the domain decomposition. In this way we create the distributed array representing the fine grid on which we perform the three dimensional FFT. We represent the fine grid arrays as a distributed ghosted vector subdivided using a mono dimensional decomposition along the coarsest variable. We need to determine two index sets representing the elements owned by a processor and its ghost elements. In this way we handle the ghost cell communications. We need two more sets to complete the necessary MPI pattern. We need to divide the input and output vectors, we do so exploiting the known FFT repartition. More precisely an element of these vectors belongs to the processor if its nearest element on the fine grid

belongs to the processor. The ghost cells are already properly set to account for this subdivision. The communications are a standard `MPI_Send`, `MPI_Recv`.

4.3.2 Implementation

We repeat the breakdown for the MPI-TBB implementation

Input gridding

We compute the first gaussian gridding from the input array to the fine grid only on those elements that are in the index set we computed following Section 4.3.1. We don't expect an optimal behavior since we are dealing with an oversampled grid, just as we explained in Section 4.2.1. If we use a high number of processors some of them will remain idle. At the end we need to communicate the elements eventually added in the ghost cells. This is a communication overhead we need to properly take into account the rest of the functions. This is a suboptimal yet very simple strategy, alternative strategies are currently being developed to prevent processors to become idle. Then we perform the correcting scaling for the first gridding. This is a pointwise operation, and we apply shared memory parallelization directly on all locally owned elements of the fine array. We apply the shift depicted in Section 4.2.1. We need to check if the elements are stored locally. At the end of the shift we communicate the values updated by the FFT on the fine grid array to the other processors in the ghost elements.

Compute 3D FFT and data shift

We simply let the underlying FFT library perform the three dimensional transformation and then we apply standard circular shift to the computed array on the locally owned elements.

Output gridding We use TBB as we did for the input, checking that each element of the output vector is on the Index set of the current processor. Finally we correct the gridding using a scaling. This is still a pointwise operation so we deal with it with a simple shared memory parallelization after we have checked that the elements are stored locally. At the end we perform a reduction to reconstruct the entire output vector starting from its distributed representation.

4.3.3 Strong scaling analysis up to 16 processors

We analyse the characteristic of the MPI parallelization alone, considering a single thread per processor, and compare it with the results obtained with TBB in Section 4.2.2. We consider the same simulation setting, forward FFT from $2^{21} = 2097152$ to $2^{21} = 2097152$ points. We plot our scalability results in Figure 4.5. For the time being we use a single computing node with an Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz processor. From Figure 4.5 we see that the most time demanding functions behave similarly. Namely we see how their slope resemble the 3D FFT one. This is due to the communication overhead and to the sub optimality of our Domain Decomposition strategy. However we see that both the shift function and the scaling on the output vector, which were two functions with optimal TBB scalability (see the left plot of Figure 4.3) present some issues. In particular both of them do not scale at all with more

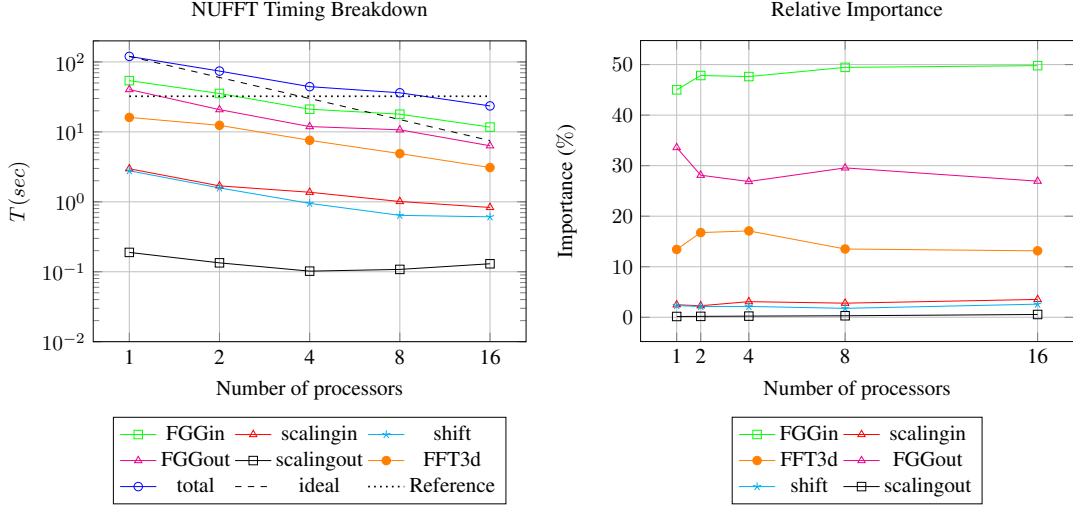


Figure 4.5: Performance analysis of the parallelization using distributed memory paradigms. We consider a single forward operation on 2097152 arbitrary points, letting the number of MPI processes vary from 1 to 16. On the left we report the actual timings of the different functions. The orange line with dots represents the 3D FFT transformation, the blue circled line is the overall scalability, the green squared one depicts the first FGG, the red triangled plot is the scaling before FFT. The starred cyan line is the shift of the data, the magenta triangled line depicts the second FGG while the black squared line is the scaling after the FFT. The black dashed line represents the optimal linear scalability. The black dotted line represents the timing of the chosen reference NUFFT library [67]. On the right we plot the relative importances of the functions needed by NUFFT for a single forward operation on 2097152 arbitrary points. The orange line with dots represents the 3D FFT transformation, the green squared one depicts the first FGG, the red triangled plot is the scaling before FFT. The starred cyan line is the shift of the data, the magenta triangled line depicts the second FGG while the black squared line is the scaling after the FFT.

than 8 processors, however we highlight that we may still use TBB to reduce their timings and that they take less than the 5 % of the overall time. We stress that, for the choices we have made, we can't expect a better overall scalability that the one of the underlying FFT library (FFTW in the present work), which is the core of our algorithm. From the comparison with the reference FORTRAN library we see that the distributed memory parallelization achieves a performance gain on the original implementation with a number of processors greater than 8.

In the right plot of Figure 4.3 we report the breakdown of the computational cost of the MPI algorithm. The most demanding functions are almost parallel straight lines, this fact evidences that our parallelization strategy has the same efficiency on all of them. We believe that a coupling of the algorithm with a parallel pruned FFT algorithm, see [92], would significantly reduce the computational cost needed by the FFT on the fine grid array. We are currently addressing this issue.

4.3.4 MPI TBB Comparison

In Sections 4.2.2 and 4.3.3 we analysed separately the two different kinds of parallelism we adopted. In the present Section we perform a comparison between the shared memory and distributed memory paradigms on the same test case scenario introduced in Section 4.2.2. We fix the overall number of parallel application (processors or threads)

and we let the number of MPI processors go from 1 to 16. This provides insights on which parallelization strategy is more convenient, varying from a pure TBB environment (1 processor, with 16 threads) to a pure MPI one (16 processors, each with 1 thread). We use a single socket composed by an Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz processor. In the left plot of Figure 4.6 we draw the timings of our algorithm. We stack all the actual timings to get a clearer representation of the different parallelization efficiencies. Looking to the overall timing we see that as we begin increasing

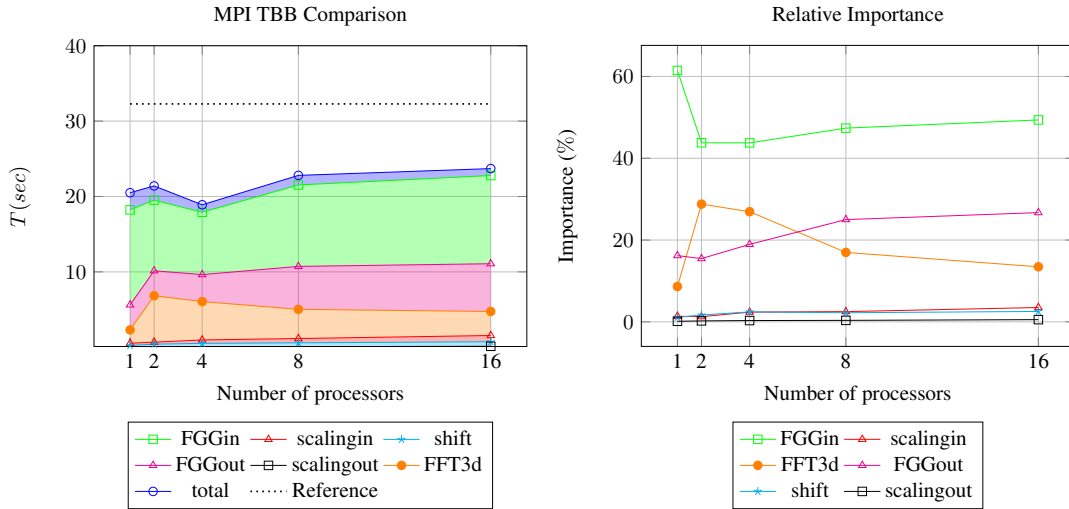


Figure 4.6: Comparison between the parallelization using different memory paradigms. memory paradigms. We consider a single forward operation on 2097152 arbitrary points. We let the number of MPI processes vary from 1 to 16 while keeping fixed to 16 the number of overall threads. On the left we draw the timings of the different parts of BlackNUFFT. The orange line with dots represents the 3D FFT transformation, the blue circled line is the overall timing, the green squared one depicts the first FGG, the red triangled plot is the scaling before FFT. The starred cyan line is the shift of the data, the magenta triangled line depicts the second FGG while the black squared line is the scaling after the FTT. The black dotted line represents the timing of the chosen reference NUFFT library [67]. On the right we plot the relative importances of the functions needed by NUFFT for a single forward operation on 2097152 arbitrary points. The orange line with dots represents the 3D FFT transformation, the green squared one depicts the first FGG, the red triangled plot is the scaling before FFT. The starred cyan line is the shift of the data, the magenta triangled line depicts the second FGG while the black squared line is the scaling after the FTT.

the number of MPI processors we experience a slight performance gain. We reach a minimum computational time when we consider 4 MPI processors with 4 threads per processor. This behavior is mainly due to the performance of the hybrid MPI-TBB parallelization of the FGG algorithm. The other functions behave almost optimally in a pure shared memory environment and this explains the slight increase of the computational time as we required too many MPI processors. In all cases we considered in the present Section our implementation has better performance than the serial reference one.

In the right plot of Figure 4.6 we draw the relative importances of all the functions varying the number of MPI processors. In this plot a decreasing or flat line means that the MPI implementation behaves better than the TBB one, since the overall time is not varying too much. From the Figure we see that only the input gridding has a

performance gain coming from the MPI parallelization. However, we highlight that the possibility of combining shared and distributed memory paradigms is of paramount importance to reach higher complexities that require more than a single computing node as described in Section 4.3.5.

4.3.5 64 bits indices compatibility

The experimental setting we have considered up to now creates a fine grid array of 373248000 complex values (746496000 actual doubles in the array). However we know that the limit of the 32 bits indexing in C++ is of 4294967296, this means that if we require more data in space or frequency we may overcome such limit. More specifically if we simply consider

$$\frac{R_{max}K_{max}}{\pi} = 200, \quad (4.6)$$

keeping all the other settings fixed, we get 5971968000 complex elements in the fine grid, and even indexing such an array becomes an issue. We provide BlackNUFFT with the possibility of using 64 bits indexing. We believe that this possibility widens the range of application of the presented library. The only requirement is that the local indexing remains confined to 32 bits indexing. This is required to properly handle local memory, and it does not seem to be a limitation because when we deal with an array of more than 2^{32} elements we unlikely use a single processor. We need to consider that an array of 5971968000 elements can be indexed using 64 bits but, if we consider double precision floating point elements, it needs 48 GigaBytes of RAM memory to be stored. Such a requirement often forces the usage of more than a single computational node and, consequently, more than a single MPI processor.

Analysis up to 32 processors In Figure 4.7 we analyse the time needed by each functions of the library to retrieve the results with the computation settings we aforementioned and that guarantee a fine grid array beyond the 32 bits limit, using 8, 16, 32 MPI processors. We use 2 nodes composed by an Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz socket. The increased dimension of the fine grid worsen the performance of the MPI parallelization, this is mostly due to the increased communication overhead. We see that FFTW maintains a good scalability with 64 bit indexing, and we also see that this is the most demanding part of the algorithm. This is induced by the increased number of communications needed by a computation with 64 bits indexing. The scaling on the output vector is not efficient at all and this is due to kind of vector we are requiring as output. We have chosen not to use distributed vector as input-output vectors, therefore we need to recreate the entire array from the distributed memory framework we are using for the NUFFT. This reconstruction requires some communication and this explains the suboptimality. A possible solution to ease the communication effort might be intranode communication if we can exploit a single computational node fulfilling the memory requirements. However we stress that in many cases we cannot require such a node, therefore we need to account for extranode communication if we need to store very big distributed arrays. From Figure 4.7 we see that the cost of the FFT becomes the major part of the overall computational cost. A possible solution to this issue is the coupling with a parallel pruned FFT algorithm as the one proposed in [92].

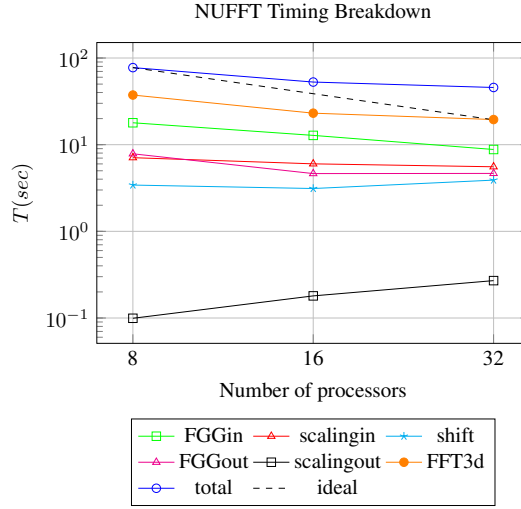


Figure 4.7: Timing of the functions needed by NUFFT for a single forward operation using a fine grid array exceeding the 32 bits limit. The orange line with dots represents the 3D FFT transformation, the blue circled line is the overall timing, the green squared one depicts the first FGG, the red triangled plot is the scaling before FFT. The starred cyan line is the shift of the data, the magenta triangled line depicts the second FGG while the black squared line is the scaling after the FTT. The black dashed line represents the optimal linear scalability.

Different job scheduling Using more than a single computational node may lead to another advantage since we are able to use more TBB threads, while keeping the number of MPI processes fixed. We analyse this case scenario comparing the timings of three different executions, exploiting the advantages of both shared and distributed memory parallelizations. We report our results in Table 4.2

Table 4.2: Performance comparison with 32 overall parallel processors. We compare three different job scheduling on 32 cores divided in two different nodes. We compute the transformation between 2097152 arbitrary points constructing a fine grid array of 5971968000 complex values. We consider two computing nodes with up to 16 processor per node. We try 2 MPI processors with 16 TBB threads each, 4 processors with 8 threads, 8 processors with 4 threads, 16 processors with 2 threads and 32 MPI processors.

Function	2 MPI 16 TBB	4 MPI 8 TBB	8 MPI 4 TBB	16 MPI 2 TBB	32 MPI 1 TBB
Total	85.1	58.2	40.2	65.7	45.7
Input Gridding	7.56	6.38	8.58	17.1	14.4
FFT	71	45.6	24.8	39	19.5
Data Shift	1.65	1.88	1.96	2.87	3.9
Output Gridding	1.36	1.44	2.32	4.13	4.93

From Table 4.2 we see that the major benefit of the pure MPI parallelization comes from the FFT routine. However we see that if we require 16 MPI processors with 2 threads per processor the performance of the FFT are worse than if we require 8 processors with 4 threads or 32 MPI processors. This is probably due to the overhead required to spawn the threads with too many MPI processors. Moreover some routines, as the gridding on the input, get advantages also from the hybrid parallelization strategy. All these different effects make the simulation with 8 MPI processors and 4 threads per

processor the most convenient from a computational point of view.

4.4 Introducing new features in the BlackNUFFT framework

This Section demonstrates how new features can be incorporated into the BlackNUFFT framework. We focus our attention on the 2 most time demanding algorithms of the library, the griddings and the three dimensional FFT. Both these options are selected through the initialising function `init_nufft` through two different string options provided by the main function.

Introducing a new gridding We briefly describes how to modify the gridding required by the NUFFT algorithm, thanks to the modularity we have provided this change can easily be achieved by providing new options to the corresponding functions.

- `compute_tolerance_infos`: since each gridding routine determines how the prescribed tolerance is used this function must be set to the specific gridding requirement. It is sufficient to implement the new case providing a new identifying string.
- `input_gridding`: it is only required to provide the new gridding function. If required it is also possible to split the routine in more functions as we did for the default Fast Gaussian Gridding scheme.
- `output_gridding`: lastly the gridding on the output vector must be modified accordingly to the new chosen gridding.

Introducing a new FFT backend library Once the new FFT library has been compiled and properly linked, or included, to the BlackNUFFT library we can easily modify the FFT depending functions using the modularity of our implementation.

- `create_index_sets`: since we have chosen to adopt the MPI redistribution of the back-end FFT library we must provide BlackNUFFT with the proper data distribution coming from the MPI requirements of the external library.
- `compute_fft_3d`: the second change required is the actual call for the backend FFT library when the fine grid array has been properly set up.

4.5 Application to fast numerical convolution: Sparse Cardinal Sine Decomposition

In this section we present a possible application of the parallel NUFFT we have developed. Boundary Integral Equations are based on the convolution of a function with a fundamental solution that is a Green like function. Fluid Structure Interaction problems profit greatly from the BIE formulation since it allows the discretization of only the boundary of the domain, which is a key advantage when large displacement are involved. The discretization procedure of BIE leads to Boundary Element Methods. Two possible examples are: ship wave interaction problem that, under certain assumptions, can be recast as a BEM for the Laplace equation, see Chapter 2 and [42, 79], and the swimming behavior of micro-organisms, that is approximated as a BEM for the Stokes

system, see Chapter 6 and [91, 109, 110]. Since the computational complexity of BEMs goes as $O(N^2)$ where N represents the number of unknowns of the problem, simulations with more than 20000 degrees of freedom are extremely time consuming even on modern workstations. Many algorithms have been developed to reduce the computational requirements to $O(N \log(N))$ or even $O(N)$. The most notable of these efforts is the Fast Multiple Method developed by Leslie Greengard in 1987 [46], for the Laplace equation. A different solution to fasten numerical convolution has been recently presented by Alouges and Aussal in [3]. The method is called the Sparse Cardinal Sine Decomposition and has shown a comparable behaviour to the classical FMM. SCSD is based on the NUFFT of type 3. Its parallelization is a key-step to obtain a parallel SCSD that exploits modern CPUs. We present a preliminary parallel implementation of the SCSD for the computation of Single and Double Layer operator of the Laplace equation, operators N, D in equation (1.9).

4.5.1 The Sparse Cardinal Sine Decomposition

We present a brief review of the mathematical aspects of the SCSD approximation. We follow [3] for the derivation of the Fast Convolution Algorithm for the Laplace equation. We focus on the approximation of two different operator that represent the convolution of the fundamental solution and its gradient.

Single Layer Operator

We evaluate the potential ϕ given by the pointwise interactions between a bunch of M sources with intensities f_j located at y_j in \mathbb{R}^3 , and N nodes at x_i . In BEMs the quantities f_j represent the density of the potential normal derivative per unit area associated to the point x_i seen as a portion of the boundary of a Lipschitz domain. If we introduce the fundamental solution of the Laplace equation as

$$G(x, y) = \frac{1}{4\pi|x - y|}, \quad (4.7)$$

we can recover the values of the potential as

$$\phi(x_i) = \sum_{j=0}^{M-1} G(x_i, y_j) f_j \quad (4.8)$$

for $i = 0, \dots, N - 1$. From standard Fourier analysis we get the following relationship for the fundamental solution

$$G(x_i, y_j) = \frac{1}{4\pi|x_i - y_j|} = \frac{1}{(2\pi)^3} \int_{\mathbb{R}^3} \frac{e^{i(x_i - y_j)\xi}}{|\xi|^2} d\xi. \quad (4.9)$$

Exploiting (4.9) we can rewrite (4.8) as

$$\phi_i = \frac{1}{(2\pi)^3} \int_{\mathbb{R}^3} e^{ix_i\xi} \sum_{j=0}^{M-1} \frac{e^{-iy_j\xi}}{|\xi|^2} f_j d\xi. \quad (4.10)$$

At this point we can pass to spherical coordinates getting

$$\phi_i = \frac{1}{(2\pi)^3} \int_0^{+\infty} \int_{S_2} e^{i\lambda x_i \zeta} \sum_{j=0}^{M-1} e^{-i\lambda y_j \zeta} f_j d\zeta d\lambda, \quad (4.11)$$

4.5. Application to fast numerical convolution: Sparse Cardinal Sine Decomposition

where S_2 stands for the sphere of radius λ . We just need two quadrature formula, one to properly approximate the spherical integrals and the other onedimensional to deal with the radial integration. For the first one we choose standard Gauss Legendre formula of arbitrary order. We apply the SCSD to retrieve an accurate onedimensional quadrature formula in λ . From standard complex analysis we know that the cardinal sine, which is an analytical function, admits the following representation,

$$\text{sinc}(x) = \frac{\sin(x)}{|x|} = \frac{1}{4\pi} \int_{S_2} e^{i\zeta x} d\zeta. \quad (4.12)$$

We can exploit the compact support of the Fourier representation of $\text{sinc}(x)$ to obtain an accurate approximation of $G(x, y)$. Combining (4.9) and (4.12) we get, defining $r = |x - y|$,

$$\frac{1}{4\pi r} = \frac{1}{2\pi^2} \int_0^{+\infty} \text{sinc}(\lambda r) d\lambda \sim \sum_p \alpha_p \text{sinc}(\lambda_p r). \quad (4.13)$$

Normalising (4.13) we get

$$1 \sim \sum_p \beta_p \sin(\lambda_p r). \quad (4.14)$$

We need to assure that, considering $x_i, y_j \in [R_{min}, R_{max}]^3 \subset \mathbb{R}^3$

$$\epsilon = \sup_{R \in [R_{min}, R_{max}]} \left| \sum_{p=0}^{p-1} \beta_p \sin((2p+1)\delta R) - 1 \right| < tol, \quad (4.15)$$

with $\delta = \frac{\pi}{R_{min} + R_{max}}$, and tol the prescribed tolerance. We consider only odd functions thus we are approximating the step function h with values 1 if $r \in]0, \pi[$ and -1 if $r \in]-\pi, 0[$. We use a Least Square approximation to retrieve a finite number of points to approximate (4.14) with the constraint (4.15). We have the following error estimate for the estimate h_P considering P terms

$$\|h - h_P\|_{L^2([\rho, \pi-\rho])} \leq \frac{1 - \sin(\rho)}{\sqrt{\sin(\rho)P}} \left(\frac{1 - \sin(\rho)}{1 + \sin(\rho)} \right)^{P+1}, \quad (4.16)$$

with $\rho \in]0, \pi/2[$. We have to stress that the approximation (4.14) leads to the usual Gibbs phenomenon, for this reason we consider the approximation correct only in $]R_{min}, R_{max}[$.

By approximating the onedimensional integral in (4.11), we can write the final discrete approximation of (4.8) as

$$\phi_i \sim \sum_{p=0}^{P-1} \omega_p \sum_{s_p=0}^{S_p-1} \omega_{s_p} e^{ix_i \xi_{s_p}} \sum_{j=0}^{M-1} e^{-iy_j \xi_{s_p}} f_j = \sum_{s=0}^{S-1} e^{ix_i \xi_s} \omega_s \sum_{j=0}^{M-1} e^{-iy_j \xi_s} f_j, \quad (4.17)$$

where ω_{s_p} represents the Gauss Legendre quadrature weights for the integral on the surface S_p and ω_p are the quadrature node coming from the cardinal sine approximation. The summation in the j index we have in (4.17) can be seen as a forward NUFFT from point y_j in space to frequencies ξ_s , while the summation in the s index is a backward NUFFT between ξ_s and x_i .

To retrieve the correct results we must apply a correction to the short range interactions, namely when $r < R_{min}$. We stress that we need to evaluate the pointwise interactions exactly, but that we can do so using a sparse matrix B ,

$$\phi = \phi_{SCSD}^{SL} + B^{SL} f, \quad (4.18)$$

where

$$B_{i,j}^{SL} = \frac{1}{4\pi|x_i - y_j|} - \sum_{p=0}^{P-1} \beta_p \frac{\sin(\lambda_p|x_i - y_j|)}{4\pi|x_i - y_j|}. \quad (4.19)$$

Double Layer Operator

We can interpret the double layer operator as a point wise interaction based on the gradient of the Green function

$$\phi_i = \sum_{j=0}^{M-1} \nabla_y(G(x_i, y_j)) \cdot g_j = \sum_{j=0}^{M-1} \sum_{k=0}^2 \nabla_y^k(G(x_i, y_j)) g_j^k. \quad (4.20)$$

where g_j represent a vectorial quantity $\in \mathbb{R}^3$. Usually in BEMs this is equal to the normal vector multiplied by the potential itself. We adapt (4.13) to the first derivative of the kernel:

$$\frac{1}{4\pi} \frac{\partial}{\partial r} \frac{1}{r} = \frac{1}{2\pi^2} \int_0^{+\infty} \frac{\partial}{\partial r} \frac{\sin(\lambda r)}{r} d\lambda \sim \sum_p \alpha_p \left(\lambda_p \frac{\cos(\lambda_p r)}{r} - \frac{\sin(\lambda_p r)}{r^2} \right). \quad (4.21)$$

Normalising (4.21) we get

$$\frac{1}{r^{2-t}} \sim \sum_p \beta_p \frac{(\sin(\lambda_p r) - \cos(\lambda_p r) \lambda_p r)}{r^{2-t}}, \quad (4.22)$$

where t is the normalising order we can set. Usually we consider $t = 1$ since this choice appears to have better approximation properties. We find a least square approximation for the radial integration as we did for the Single Layer, and we retrieve the SCSD approximation as

$$\phi_i \sim \sum_{s=0}^{S-1} e^{ix_i \xi_s} \omega_s \sum_{k=0}^2 i \xi_s^k \sum_{j=0}^{M-1} e^{-iy_j \xi_s} g_j^k, \quad (4.23)$$

where we have taken the spatial derivatives simply multiplying by the three different Fourier constants. We just need a final adjustment for the short range correction, namely,

$$\phi = \phi_{SCSD}^{DL} + \sum_{k=0}^2 B^{DL_k} g^k, \quad (4.24)$$

where

$$B_{i,j}^{DL_k} = \frac{x_i^k - y_j^k}{4\pi|x_i - y_j|^3} - \sum_p \beta_p \frac{(\sin(\lambda_p r) - \cos(\lambda_p r) \lambda_p r)}{r^2}. \quad (4.25)$$

4.5. Application to fast numerical convolution: Sparse Cardinal Sine Decomposition

4.5.2 Algorithm key-steps

In this section we sum up the essential steps we need in order to perform a SCSD convolution.

- **Geometry Setting:** we set R_{max} which is the maximum distance between our points and R_{min} that determines the accuracy range of the SCSD approximation. Decreasing the latter value increases the number of spheres we need to approximate the Fourier integral.
- **Onedimensional quadratures computation:** we determine weight of the quadrature formula coming from the SCSD approximation, namely ω_p in (4.17). This computation sets all the radii we need to approximate the integral in (4.11).
- **Three dimensional quadrature for the frequency domain:** for each radius we have previously determined using the mono dimensional approximation we need a proper quadrature formula on the sphere. We use standard Gauss Legendre formulas of arbitrary order, and satisfy the prescribed tolerance for each sphere.
- **Short range Interaction setting:** for every node x_i we use R_{min} to determine which sources y_j are such that $|x_i - y_j| < R_{min}$. We need their index and the distance to determine the short range correction matrices.
- **Fast convolution using NUFFT:** we apply (4.17) and (4.23) to approximate Single and Double Layer Operators respectively.
- **Short range interaction correction:** lastly we apply (4.18) and (4.24) to get the proper short range interactions. We use *nanoflann* [17] to identify such short range interactions. This operation is basically a sparse matrix vector multiplication.

4.5.3 Parallelization strategies

As we did for the NUFFT library in Sections 4.2 and 4.3 we analyze two different parallelization, a pure multithread approach using TBB and then a hybrid parallelization coupling TBB and MPI. The only part of the algorithm we do not parallelize at all is the onedimensional computation of the quadrature formula. This is not an issue since it is a simple onedimensional computation and it is cheaper to replicate it on each processor.

TBB

We use TBB, as described in [115], to parallelize every section of the SCSD algorithm, Section 4.2 shows that the developed BlackNUFFT do not scale perfectly, therefore we can't expect optimum results for the SCSD-NUFFT part. We can, however, exploit the embarrassingly parallel structure of (4.8) and (4.20) to get good results for all the other parts of the algorithm.

We consider a pointwise interaction between two clouds of random points x_i and y_j located in $[-1, 1]^3$ and we set $N = M = 100000$. We analyze both Single and Double Layer operators to have a complete evaluation of the first parallelization technique, we consider $R_{min} = 0.1$. We expect the computations of the Double Layer operator to be more expensive (from a timing point of view) than the ones of the Single Layer operator. We perform a strong scaling analysis using up to 16 threads on a computing node

Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz. We report the timing analysis in Figure 4.8. We see that the short range computations, green squarred and triangled lines,

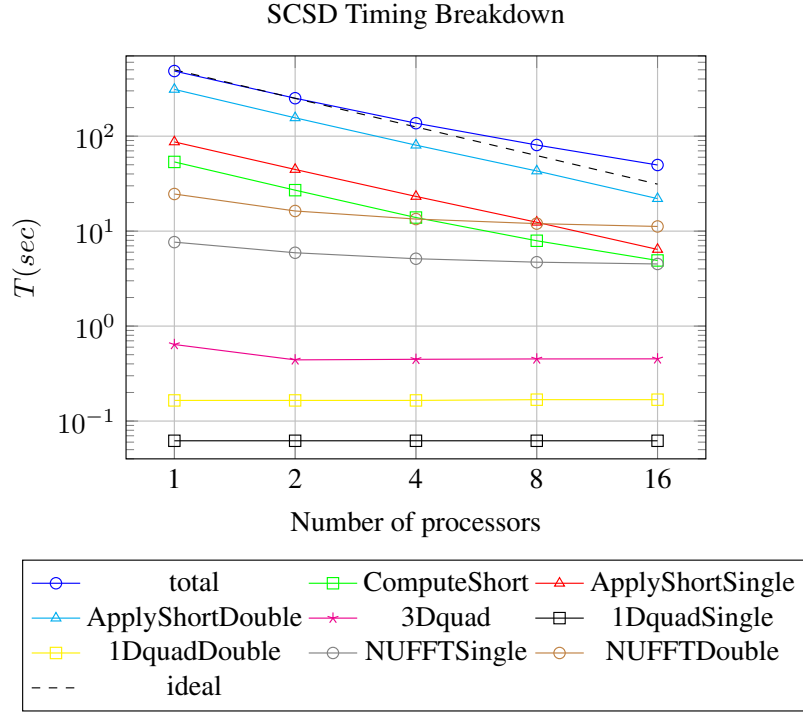


Figure 4.8: Timing Analysis for a single SCSD computation for both the Single Layer and Double Layer Operators using only multithread parallelization techniques. The blue circled line represents the total timing, the green squarred one describes the computation of short range interactions, the red and cyan triangled lines show the actual application of such short range effects for the Single and Double Layer operators. The magenta starred line plots the time needed to set up the 3D quadrature formula in the Fourier space while yellow and black squarred lines represent the timings required for the 1D quadrature formula computation using SCSD for Single and Double Layer operators respectively. Finally gray and brown circled plots represent the computational time required by the NUFFT algorithm for Single and Double Layer operators. We plot the ideal timings using a black dashed line.

present an almost optimal scalability while the NUFFT parts, brown and black circled plots, do not perform optimally. Since the most computational expensive functions are the short range computations we see almost an optimal scaling up to 8 threads. Then we see that the bottleneck becomes the NUFFT computations that don't show such an optimal behavior. In Section 4.3 we saw that a hybrid parallelization is the optimal choice for the NUFFT algorithm, so we look for this kind of implementation even for the complete SCSD.

Hybrid MPI-TBB

We apply a distributed memory parallelization also to the overall SCSD algorithm. The key-steps of the MPI strategy are the following

- **MPI Repartition of the output vector:** we subdivide the vector dividing its elements between all the processors, we also handle the remainder by means of a subdivision among the first processors.

4.5. Application to fast numerical convolution: Sparse Cardinal Sine Decomposition

- Short Range Interaction: we let each processor compute only the ranges for its owned elements of the output processor.
- SCSD with BlackNUFFT: we let the library compute the interactions exploiting the parallelization we have introduced in Section 4.3.
- Apply Short Range Interaction: each processor compute the corrections only for its owned elements of the output vector.
- Sum of the contributions: we sum up the short range corrections to the output vector.

As benchmark we consider the same pointwise interaction between of the previous Section (two clouds of random points x_i and y_j located in $[-1, 1]^3$ and $N = M = 100000$). We use a single computation node, and we use up to 16 processor using a single thread per processor, highlighting the effects of a pure MPI parallelization. We report the analysis in Figure 4.9. We see that the performance of a the MPI parallelization are very

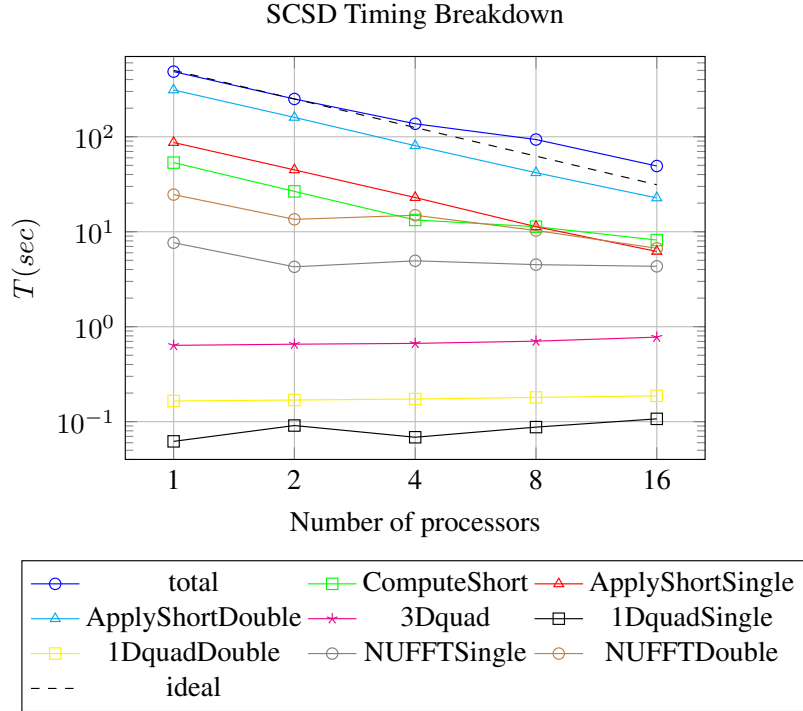


Figure 4.9: Timing Analysis for a single SCSD computation for both the Single Layer and Double Layer Operators using only hybrid distributed memory parallelization techniques. The blue circled line represents the total timing, the green squared one describes the computation of short range interactions, the red and cyan triangled lines show the actual application of such short range effects for the Single and Double Layer operators. The magenta starred line plots the time needed to set up the 3D quadrature formula in the Fourier space while yellow and black squared lines represent the timings required for the 1D quadrature formula computation using SCSD for Single and Double Layer operators respectively. Finally gray and brown circled plots represent the computational time required by the NUFFT algorithm for Single and Double Layer operators. We plot the ideal timings using a black dashed line.

similar to the ones reported in Figure 4.8. In particular the correction due to the embarrassingly parallel short range interactions has an optimal behavior. While the NUFFT

parts behave similarly using TBB or MPI, we see that the computations of the quadrature formulas is requiring more time as we increase the number of MPI processors. This could lead to new bottlenecks, for Amdhal's law, if we increase the computational requirements of the simulation. However, we stress that TBB can easily be applied to the computation of the three dimensional Fourier grid, sub-optimally as we can see from Figure 4.8. A MPI parallelization to this part is far from trivial, it would involve a MPI broadcast for every sphere. For the time being we have not been able to apply successfully such a strategy.

The best option is a hybrid parallelization that should reduce the effects of Amdhal's law to the minimum. In Figure 4.10 we report a comparison between the two strategies. We always require the maximum of processors to be 16 and we increase the number of MPI processors up to 16. We report the actual timings for the main functions of the SCSD algorithm.

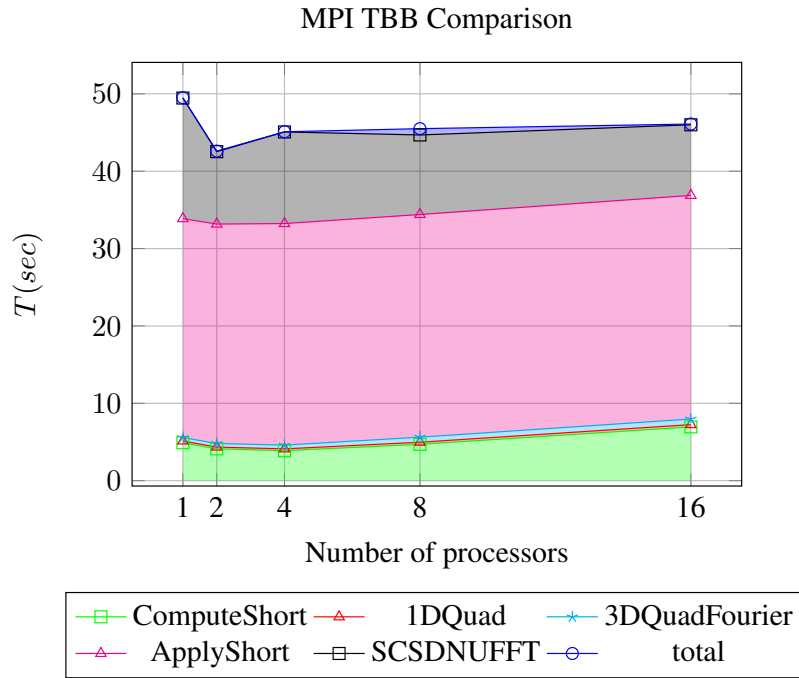


Figure 4.10: MPI TBB comparison for a single SCSD computation for the Single Layer Operator. The blue circled line is the overall time, the green squared one depicts the computation of the short range interactions, the red triangled one is the 1D Least Square minimisation. The starred cyan represents the time needed by the 3D quadrature formula in Fourier space. The magenta triangled line depicts the application of the short range interactions while the black squared line is the timing of the NUFFT part.

We see a minimum if we consider 2 MPI processors and 8 threads per processors. This minimum is induced by the NUFFT part of the algorithm that benefits from a hybrid parallelization. The application of short range interactions has the same performance in MPI and TBB, so the relative timing does not change significantly. We see that the computations of the short range interactions worsen the performance of the overall algorithm if we require more than 4 MPI processors. This is a clear example of Amdhal's law in action: since this part is not properly parallelized in MPI the overall scalability is compromised. We believe the hybrid memory parallelism to be the

4.5. Application to fast numerical convolution: Sparse Cardinal Sine Decomposition

most effective one, the study of a proper MPI parallelization both of the short range computations and three dimensional quadrature formula is mandatory to improve the performance of the code.

The NUFFT parts do not present good performance for the presented case. We highlight that we are requiring only 100000 points while in Section 4.3 we used up to 2097152 elements. We believe that different choices of the initial configuration may lead to very different scalability results, we are studying these differences analyzing the input-output points position x_i, y_j , and quantities N, M . In particular, we are interested in the performance of the algorithm when these points are placed on a surface, such a configuration would allow the coupling of the SCSD algorithm to the BEM presented in Section 3. We are also analyzing the sensitivity of the problem depending on the R_{min} parameter. It manages the workbalance between the short range interactions and the NUFFT part: namely as R_{min} decreases the NUFFT part becomes more relevant on the overall computation, therefore an optimal balance between NUFFT and short range computations can be obtained varying R_{min} .

CHAPTER 5

Stokes Boundary Integral Formulation

We discuss the derivation of a Boundary Element Method for the Stokes system. Such mathematical setting describes an incompressible Newtonian fluid in which inertial terms can be neglected. Given the peculiarities of the Navier–Stokes equations such approximation holds both for extremely slow flows (creeping flows) and for extremely small geometry $O(10^{-6}m)$ (micro fluid-dynamics). The only requirement in the derivation of a BEM is the explicit knowledge of a fundamental (Green-like) solution. The Stokes system is a second order elliptic problem, and it admits a well known fundamental solution. It is possible to modify the standard fundamental solution to model different physical interfaces as a single or a system of perfect slip walls. Such interfaces are of paramount importance to simulate real life swimmers since they deeply influence the flow around a moving body. Their actual numerical discretization is very expensive from a computational point of view, the derivation of specific fundamental solutions reduces the number of unknowns N easing the computational requirement of the simulation.

This Chapter is organized as follows, in Section 5.1 we introduce the Boundary Integral Formulation for the Stokes system, in Section 5.2 we discuss the numerical implementation and requirement for a BEM in the Stokes framework, and in Section 5.3.1 we derive a modified fundamental solution to model an infinite perfect slip interface using the reflection method. In Section 5.3.2 we propose an approximation for a Green solution modeling a flow through two infinite perfect slip planes.

5.1 Boundary Integral Formulation

We study the flow past an obstacle (a “swimmer”) seen as a bounded open set $B_t \subset \mathbb{R}^d$, with Lipschitz boundary ∂B_t . We call the fluid domain Ω , having boundary $\Gamma = \partial\Omega$.

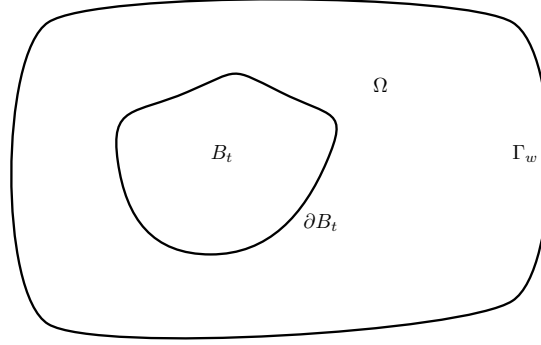


Figure 5.1: General sketch of the considered domain Ω with $\partial\Omega = \Gamma$: for a free space swimmer $\Gamma = \partial B_t$ and $\Omega = \mathbb{R}^d \setminus \bar{B}_t$, for a swimmer near no slip walls $\Gamma = \partial B_t \cup \Gamma_w$.

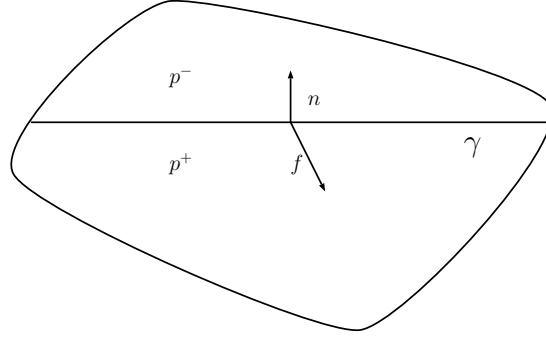


Figure 5.2: Representation of the stresses in a fluid portion. f is the force per unit area acting on γ .

Both free space and confined swimming are of interest in this work. In the first case $\Omega = \mathbb{R}^d \setminus \bar{B}_t$ and $\Gamma = \partial B_t$, while in the second case, we indicate with Γ_w the physical walls and $\Gamma = \partial B_t \cup \Gamma_w$. We sketch the considered domain in Figure 5.1. We consider an incompressible Newtonian fluid, consequently we define the stress f , see Figure 5.2 as the force per unit area that the portion of fluid in p^+ exerts on the portion of fluid in p^- through the surface γ . We call the unit outer normal vector with respect to p^+ $n(x)$. The stress, from classical continuum mechanics [51], at a point x (on γ) is given by:

$$f(x) = \sigma(u(x), p(x))n(x), \quad (5.1)$$

where n and $\sigma(u, p)$ represent the normal vector and the Cauchy stress tensor respectively, the latter reads

$$\sigma(u, p) = 2\mu \left(\frac{1}{2} \nabla u + \frac{1}{2} \nabla u^T \right) - pI, \quad (5.2)$$

see [51] for further details. The Stokes system reads

$$\nabla \cdot u = 0 \text{ in } \Omega \quad (5.3a)$$

$$\Delta u + \nabla p = 0 \text{ in } \Omega. \quad (5.3b)$$

We follow [19,93] to retrieve a Boundary Integral Formulation starting from (5.3). We consider the full representation formula for the velocity in (5.4)

$$u_i(x) - \int_{\Gamma} T_{ijk}(x, y) n_k(y) u_j(y) d\gamma_y = \int_{\Gamma} G_{ij}(x, y) f_j(y) d\gamma_y \quad \forall x \in \mathbb{R}^d \setminus \Gamma, \quad (5.4)$$

where G, T represents the first two Green tensors associated with the fundamental solution of (6.11), namely solving

$$\nabla \cdot Gb = 0 \quad (5.5a)$$

$$\nabla \cdot (\sigma(Gb)) = \delta b, \quad (5.5b)$$

$$T_j = \sigma(G_j), \quad (5.5c)$$

where δ is the Dirac delta centered at the origin, G_j represents the column of G associated to the forcing term δe_j and b is a generic vector. If we define $r = x - y$ and $R = |x - y|$, the free space expressions of G, T are as follows, see [113],

$$\begin{cases} G_{ij} = \frac{1}{4\pi} \left(\frac{r_i r_j}{R^2} - \delta_{ij} \log(R) \right) & \text{in 2D} \\ G_{ij} = \frac{1}{8\pi} \left(\frac{r_i r_j}{R^3} + \delta_{ij} \frac{1}{R} \right) & \text{in 3D,} \end{cases} \quad (5.6)$$

$$\begin{cases} T_{ijk} = -\frac{1}{\pi} \frac{r_i r_j r_k}{R^4} & \text{in 2D} \\ T_{ijk} = -\frac{3}{4\pi} \frac{r_i r_j r_k}{R^5} & \text{in 3D.} \end{cases} \quad (5.7)$$

We consider the trace of (5.4) to compute the real Boundary Integral Equation of the Stokes system,

$$\alpha(x)u_i(x) - \int_{\Gamma}^{PV} T_{ijk}(x, y)n_k(y)u_j(y)d\gamma_y = \int_{\Gamma} G_{ij}(x, y)f_j(y)d\gamma_y \quad \forall x \in \Gamma, \quad (5.8)$$

where the integral on the left are computed in the Cauchy principal value sense, and α represents its Cauchy principal value. Equation (5.8) consists in two boundary integral operators

$$Hu = \int_{\Gamma}^{PV} T_{ijk}(x, y)n_k(y)u_j(y)d\gamma_y, \quad (5.9a)$$

$$Vf = \int_{\Gamma} G_{ij}(x, y)f_j(y)d\gamma_y, \quad (5.9b)$$

where H is called double layer operator and V single layer operator. We introduce the operator notation of (5.8) as

$$[\alpha I - H]u = -[K]u = -[V]f. \quad (5.10)$$

For the integrals in equation (5.8) to be bounded, u and f must lie in the spaces V and Q , defined as

$$V := \left\{ \phi \in (H^{\frac{1}{2}}(\Gamma))^d \right\} \quad (5.11a)$$

$$Q := \left\{ \gamma \in (H^{-\frac{1}{2}}(\Gamma))^d \right\}, \quad (5.11b)$$

where $\Gamma = \partial\Omega$ and $n = 2, 3$. We recall that $(H^{\frac{1}{2}}(\Gamma))^d$ is usually defined as the space of traces on Γ of functions in $(H^1(\Omega))^d$, while $(H^{-\frac{1}{2}}(\Gamma))^d$ is its dual space. The spaces V_h and Q_h are constructed as conforming finite dimensional subspaces of V and Q respectively. From a functional point of view we define the double and single layer operator as

$$H : (H^{\frac{1}{2}}(\Gamma))^d \rightarrow (H^{\frac{1}{2}}(\Gamma))^d, \quad (5.12a)$$

$$V : (H^{-\frac{1}{2}}(\Gamma))^d \rightarrow (H^{\frac{1}{2}}(\Gamma))^d. \quad (5.12b)$$

5.2 Stokes Boundary Element Method

We closely follow the implementation of Laplace BEM reported in Section 1.2, and we apply the parallelization principles described in Chapter 3. We base our implementation on the use of existing high performance libraries as `deal.II` [6] and Trilinos [57]. With the Boundary Integral Equation (5.8) it is possible to retrieve the velocity where the stress are known, and viceversa. A discretization of the approximation spaces and of the boundary are the only requirements. We use standard Lagrangian finite element spaces on Γ to define both the geometry and the basis functions for the unknowns, namely the velocity u and the stresses f . We provide the possibility of using both continuous and discontinuous approximation for the solution. As we did for the Laplace equation in Section 1.2 we divide the finite dimensional approximation of the problem in 4 main steps.

Computational mesh generation We define a quadrilateral computational mesh meant as a regular decomposition Γ_h of the boundary Γ . See Section 1.2 for further details. The integration of complex geometrical models as CAD files, as we did in Section 1.3.4, to describe the swimmer is currently under development.

Definition of the discrete spaces We define two finite dimensional spaces V_h and Q_h , for the Stokes system we need to handle the vectorial nature of the problem. The discrete spaces needs to be defined on Γ_h , such that

$$V_h := \{ \phi_h \in (L^2(\Gamma_h))^d : \phi_{h|K} \in (\mathbb{Q}^r(K))^d, K \in \Gamma_h \} \equiv \text{span}\{ \psi_i \}_{i=1}^{dN_V} \quad (5.13a)$$

$$Q_h := \{ \gamma_h \in (L^2(\Gamma_h))^d : \gamma_{h|K} \in (\mathbb{Q}^s(K))^d, K \in \Gamma_h \} \equiv \text{span}\{ \omega_i \}_{i=1}^{dN_Q}, \quad (5.13b)$$

where on each cell K , located on the boundary, $\phi_{h|K}, \gamma_{h|K}$ are polynomial functions of degree r and s respectively, in each coordinate direction and for any component of the vectorial problem. In principle these two spaces can be built independently, but for the moment we restrict the discussion to the case where we use the same Finite Element discretisation for both the primal and the dual unknown, i.e., $V_h = Q_h \equiv \text{span}\{ \psi_i \}_{i=1}^{dN_V}$ for any component of the vectorial problem, thus we can write the finite dimensional approximations for velocity and stresses as

$$u_h = \sum_{i=1}^{dN_V} \psi_i(x) \hat{u}_i \quad (5.14a)$$

$$f_h = \sum_{i=1}^{dN_V} \psi_i(x) \hat{f}_i, \quad (5.14b)$$

where \hat{u}, \hat{f} represent the nodal values for the velocity and the stresses respectively. We use continuous Lagrangian final elements to accurately describe the geometry, afterwards we apply a collocation method on the support points of the geometry patches directly on the specified geometry.

Collocation of the Boundary Integral Equations As we stated in Section 1.2 in BEM framework the collocation method appears as a natural choice since it avoids a second integration of weakly singular kernels. We replace the continuous functions u and f with their numerical approximations u_h and f_h , which represent the discretised velocity and stress density respectively in the finite dimensional space, and we collocate the BIE on a number of points equal to the number of unknowns. We choose, as collocation points, the support points of the basis functions on the boundary Γ_h .

Collocating (5.8) produces the linear system

$$(\alpha I_h - H_h)\hat{u} = -K_h\hat{u} = -V_h\hat{f} = 0, \quad (5.15)$$

where

- α is a $d \times d$ block diagonal matrix representing the Cauchy Principle Value of the double layer operator H_h , $\alpha(x_i)$, where x_i represents the i -th collocation point;
- $H_{ij} = \sum_{k=1}^{N_k} \sum_q^{N_q} T_{itp}(x_i - x_q) n_k(x_q) \psi_t^j(x_q) J^k$, where k represents the reference cell and J^k is the determinant of the first fundamental form for each panel k ;
- $V_{ij} = \sum_{k=1}^{N_k} \sum_q^{N_q} G_{it}(x_i - x_q) \psi_t^j(x_q) J^k$;
- \hat{u}, \hat{f} represent the nodal value of the velocity vector field and of the stress tensor field respectively.

As in Section 1.2 the integrals on the reference cell are performed using different quadrature schemes. When the collocation point does not lie inside the cell we simply use bidimensional Gauss scheme, otherwise we need a different strategy since the Kernels G, T are weakly singular, see [114]. On one hand this choices allows for an accurate solution of the weakly singular Boundary Integral Equation, and on the other hand it makes the assemblage cycle more complicated. This is mainly due to the fact that we need to use two different discrete schemes and to check if the collocation point lies inside the current cell.

Resolution of the linear system Depending on the problem we derive the system matrix A and we solve it by means of a preconditioned Krylov iterative solver or using a direct parallel linear solver. We discuss the detail of the swimming problem, a possible application of such Boundary Element Method, in Section 6.2.

5.3 Kernels for specific interfaces

In the previous Sections we saw that a proper discretization of the boundary is mandatory to develop a reliable Boundary Element Method, therefore, in general, when we consider a physical interface Γ_w we need to provide a proper finite dimensional representation of that surface. However, in some cases it is possible to modify the kernels G, T appearing in (5.9) to automatically satisfy the required boundary conditions on Γ_w .

This technique is particularly suitable for two main reasons: since it does not require a discretization of Γ_w it reduces the overall number of degrees of freedom (reducing the

computational time), and it naturally treats infinite interfaces. However such methodology can be applied only to certain geometries having simple (either perfect slip or no slip) boundary conditions. Section 5.3.1 presents the derivation of kernels G^{PS}, T^{PS} to model a single flat infinite perfect slip interface. These kernels are used to simulate the interactions of the bacterium with the wall in [91], we apply such fundamental solutions to verify the reliability of the presented BEM in Section 6.4.4.

In many cases, however, more complex interfaces are necessary. It is common to use a thin soap film as experimental setting, this reduces the swimmer movements out of the plane of focus of the camera. In [50] the following setup is used: the swimmers (*Chlamydomonas reinhardtii*) are $10 - 20\mu m$ long, the film is square with side length $6mm$ and measured thickness $15 \pm 2\mu m$. For this scenario the expressions of the modified kernels are still not known, in Section 5.3.2 we propose a possible approximation of these fundamental solutions to simulate the flow past two infinite perfect slip interfaces. Section 6.5 proposes an application of the derived approximated kernel to the experimental setting proposed in [50].

5.3.1 Single perfect slip flat interface

We analyze the procedure to derive a kernel for the Stokes equation to model the presence of a single perfect slip interface. We model the two green kernels

- G , the tensor representing the fundamental solution along three orthonormal vectors,
- T , the tensor representing the three stresses associated to the three fundamental solutions.

Perfect slip conditions and mirroring

The solution of the stokes problem u fulfills

$$u_{wall} \cdot n = V_n|_{wall} = 0, \quad (5.16a)$$

$$\sigma(u)n|_{wall} \cdot t = 0. \quad (5.16b)$$

An equivalent way to fulfill equations (5.16) is to mirror the flow with respect to the plane representing a perfect slip interface. For the sake of simplicity we consider the case of the (x, z) plane at $y = 0$. Mirroring implies:

- $V_n^+ = -V_n^-$;
- $V_t^+ = V_t^-$.

From symmetry considerations (see Figure 5.3) we get

- $V_n|_{wall} = 0$,
- $\frac{\partial V_t}{\partial n}|_{wall} = 0$,
- $\frac{\partial V_n}{\partial n}|_{wall} = \frac{\partial V_n^+}{\partial n}|_{wall} = C_1$,

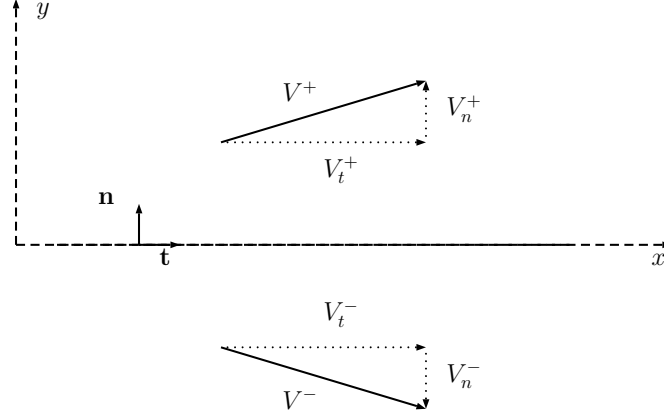


Figure 5.3: Mirroring of the flow modeling a semi-infinite flat perfect slip surface.

- $\frac{\partial V_t}{\partial t} \Big|_{wall} = \frac{\partial V_t^+}{\partial t} \Big|_{wall} = C_2,$
- $\frac{\partial V_n}{\partial t} \Big|_{wall} = 0.$

So the gradient of the velocity $\nabla u|_{wall}$ reads

$$\nabla u|_{wall} = \begin{bmatrix} \frac{\partial V_n}{\partial n} & \frac{\partial V_n}{\partial t} \\ \frac{\partial V_t}{\partial n} & \frac{\partial V_t}{\partial t} \end{bmatrix} = \begin{bmatrix} C_1 & 0 \\ 0 & C_2 \end{bmatrix}, \quad (5.17)$$

and we get

$$\sigma(u)|_{wall} = \begin{bmatrix} C_1 + p & 0 \\ 0 & C_2 + p \end{bmatrix}. \quad (5.18)$$

From expression (5.18), we obtain the second condition of the perfect slip flow, *i.e.*:

$$\sigma n|_{wall} \cdot t|_{wall} = \begin{bmatrix} C_1 + p & 0 \\ 0 & C_2 + p \end{bmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 0, \quad (5.19)$$

Equation (5.19) proves that the mirroring of the solution is equivalent to perfect slip conditions.

Kernel derivation using geometric considerations

We define the fundamental solution modeling a perfect slip surface: G^{PS}, T^{PS} . We exploit the properties of the standard Green function for the Stokes problem to retrieve a representation formula for the velocity u

$$u(x)_i - \int_{\Gamma} T_{ijk}(x, y) n_k(y) u_j(y) d\gamma_y = \int_{\Gamma} G_{ij}(x, y) f_j(y) d\gamma_y \quad \forall x \in \mathbb{R}^d \setminus \Gamma. \quad (5.20)$$

Following the considerations of Section 5.3.1, if we consider a body moving near a free slip surface we can equivalently consider the motion of the body itself and its mirrored image. We have depicted the setting in Figure 5.4 .

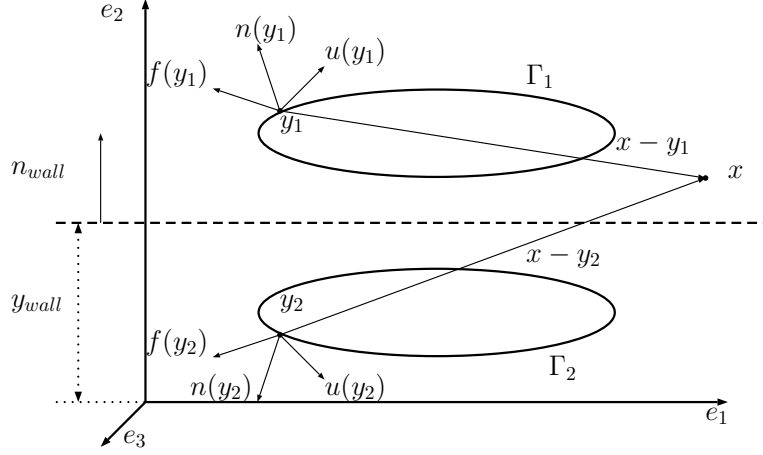


Figure 5.4: The mirroring of the bodies depicted in equation (5.21). We see the real boundary of the body described as Γ_1 and its mirror image Γ_2 . The perfect slip interface is perpendicular to e_2 .

Namely we write

$$u(x)_i - \int_{\Gamma_1} T_{ijk}(x, y_1) n_k(y_1) u_j(y_1) d\gamma_{y_1} - \int_{\Gamma_2} T_{ijk}(x, y_2) n_k(y_2) u_j(y_2) d\gamma_{y_2} = \int_{\Gamma_1} G_{ij}(x, y_1) f_j(y_1) d\gamma_{y_1} + \int_{\Gamma_2} G_{ij}(x, y_2) f_j(y_2) d\gamma_{y_2} \quad \forall x \in \mathbb{R}^d \setminus \Gamma. \quad (5.21)$$

We define G^{PS}, T^{PS} to be the two tensors satisfying the following equations

$$\int_{\Gamma_1} G_{ij}^{PS}(x, y_1) f_j(y_1) d\gamma_{y_1} = \int_{\Gamma_1} G_{ij}(x, y_1) f_j(y_1) d\gamma_{y_1} + \int_{\Gamma_2} G_{ij}(x, y_2) f_j(y_2) d\gamma_{y_2} \quad (5.22a)$$

$$\int_{\Gamma_1} T_{ijk}^{PS}(x, y_1) n_k(y_1) u_j(y_1) d\gamma_{y_1} = \int_{\Gamma_1} T_{ijk}(x, y_1) n_k(y_1) u_j(y_1) d\gamma_{y_1} + \int_{\Gamma_2} T_{ijk}(x, y_2) n_k(y_2) u_j(y_2) d\gamma_{y_2} \quad (5.22b)$$

If equations (5.22a) and (5.22b) are satisfied we can write

$$u(x)_i - \int_{\Gamma} T_{ijk}^{PS}(x, y) n_k(y) u_j(y) d\gamma_y = \int_{\Gamma} G_{ij}^{PS}(x, y) f_j(y) d\gamma_y \quad \forall x \in \mathbb{R}^d \setminus \Gamma, \quad (5.23)$$

(5.23) is equivalent to (5.21). We consider the symmetries of the vector fields n, u, f , and we define

$$y_2 = y_1 - 2n_{wall}(n_{wall} \cdot y_1 - y_{wall}), \quad (5.24a)$$

$$f(y_2) = Af(y_1), \quad (5.24b)$$

$$u(y_2) = Au(y_1), \quad (5.24c)$$

$$n(y_2) = An(y_1), \quad (5.24d)$$

where y_{wall} specifies the wall location, and A is a tensor specifying the symmetry relations, namely

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (5.25)$$

The Green functions G, T depends on the difference between the source and the evaluation point, namely $R = x - y$. We have the following relations between $R_1 = x - y_1 = (r_x^1, r_y^1, r_z^1)$ and $R_2 = x - y_2 = (r_x^2, r_y^2, r_z^2)$,

$$r_x^2 = r_x^1, \quad (5.26a)$$

$$r_y^2 = r_y^1 - 2(y - y_{wall}) \quad (5.26b)$$

$$r_z^2 = r_z^1. \quad (5.26c)$$

We introduce the modified Green tensor $G^{PS}(R_1) = G(R_1) + \tilde{G}(R_2)$ with

$$G_{ij} = \frac{1}{8\pi} \left(\frac{\delta_{ij}}{R_1} + \frac{r_i^1 r_j^1}{R_1^3} \right), \quad (5.27)$$

$$\tilde{G}_{ij} = \begin{cases} \frac{1}{8\pi} \left(\frac{\delta_{ij}}{R_2} + \frac{r_i^2 r_j^2}{R_2^3} \right) & \text{if } j \neq 2 \\ -\frac{1}{8\pi} \left(\frac{\delta_{ij}}{R_2} - \frac{r_i^2 r_j^2}{R_2^3} \right) & \text{if } j = 2 \end{cases} \quad (5.28)$$

this subdivision is needed to ensure the mirroring of forces parallel to n_{wall} as it appears from (5.24b). We derive an expression for the second kernel $T^{PS} = T(R_1) + \tilde{T}(R_2)$, with

$$T_{ijk} = -\frac{3}{4\pi} \left(\frac{r_i^1 r_j^1 r_k^1}{R_1^5} \right), \quad (5.29)$$

$$\tilde{T}_{ijk} = \begin{cases} -\frac{3}{4\pi} \left(\frac{r_i^2 r_j^2 r_k^2}{R_2^5} \right) & \text{if } j \neq 2 \text{ and } k \neq 2 \\ +\frac{3}{4\pi} \left(\frac{r_i^2 r_j^2 r_k^2}{R_2^5} \right) & \text{if } j \neq 2 \text{ and } k = 2 \\ +\frac{3}{4\pi} \left(\frac{r_i^2 r_j^2 r_k^2}{R_2^5} \right) & \text{if } j = 2 \text{ and } k \neq 2 \\ -\frac{3}{4\pi} \left(\frac{r_i^2 r_j^2 r_k^2}{R_2^5} \right) & \text{if } j = 2 \text{ and } k = 2 \end{cases} \quad (5.30)$$

We introduced a double reflection in (5.30) to take consider both (5.24c) and (5.24d). We examine the two tensors G^{PS}, W^{PS} considering x on the mirroring plane.

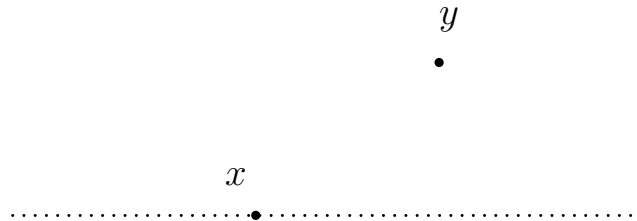


Figure 5.5: Test point x on the symmetry plane.

We consider the following setting $y_{wall} = 3.5$, the body is a sphere of radius equal 1 placed at $C = (0, 0, 0)$. We tow the sphere near the perfect slip interface, modeled using (5.28) and (5.30). We expect the perfect slip condition (5.16a) (5.16b) to be fulfilled. Since we evaluate the velocity by means of equation (5.23) on a point of the symmetry plane we automatically get $G_{2j}^{PS} = 0, T_{2jk}^{PS} = 0$, since we have that $r_1^1 = r_1^2, r_2^1 = -r_2^2, r_3^1 = r_3^2$. We expect the vertical velocity $u_2(x)$ to vanish $\forall x \in wall$. We consider two points on the wall and compute the velocity

$$\begin{aligned} x = (10, 3.5, 10), u &= (0.00803196, -8.90162e^{-22}, 0.00256115), \quad u_2 = -8.90162e^{-22}, \\ x = (-3, 3.5, -5), u &= (0.0140131, -9.99815e^{-22}, 0.00371258), \quad u_2 = -9.99815e^{-22}. \end{aligned}$$

Then we compute ∇u to verify (5.16b) as well. We get

$$\begin{aligned} x = (10, 3.5, 10), \nabla u_1 &= (-0.00021436, 0, -0.00123881), \\ \nabla u_2 &= (-4.88837e^{-16}, 0.000423812, -1.02174e^{-16}) \\ \nabla u_3 &= (-0.0002094620 - 0.000209452), \\ x = (-3, 3.5, -5), \nabla u_1 &= (-0.000565304, 0, 0.004008), \\ \nabla u_2 &= (-7.31933e^{-15}, -0.000322313, -1.28463e^{-15}) \\ \nabla u_3 &= (-0.00105157, -4.33681e^{-13}, 0.000887617). \end{aligned}$$

The symmetric formulations (5.28), (5.30) automatically satisfy the perfect slip boundary conditions (5.16a), (5.16b).

Kernel derivation using Lorentz identity

We derive the kernels starting from the Lorentz identity [93]. We consider the well known Lorentz reciprocal identity, given two different solutions of the Stokes system u, u' with the associated stress tensors σ, σ' . We have that

$$\frac{\partial}{\partial x_j} (u'_i \sigma_{ij} - u_i \sigma'_{ij}) = u'_i \frac{\partial \sigma_{ij}}{\partial x_j} - u_i \frac{\partial \sigma'_{ij}}{\partial x_j}. \quad (5.31)$$

If we consider

$$u_i(x) = G_{ij}(x, x_1) b_j \quad (5.32a)$$

$$\frac{\partial \sigma_{ij}}{\partial x_j} = \delta(x - x_1) b_j \quad (5.32b)$$

$$u'_i(x) = G_{ij}(x, x_2) a_j \quad (5.32c)$$

$$\frac{\partial \sigma'_{ij}}{\partial x_j} = \delta(x - x_2) a_j, \quad (5.32d)$$

the following property for the Green function $G_{i,j}(x_1, x_2)$ must hold:

$$G_{ij}(x_1, x_2) = G_{ji}(x_2, x_1). \quad (5.33)$$

We define the stress tensor associated to $G(x, x_1)$ as

$$T_{ijk}(x, x_1) = \sigma_x(G_{ij}(x, x_1)). \quad (5.34)$$

Where x in the subscript indicates that we are taking all differential operators with respect to the variable x . The following must be fulfilled

$$T_{ijk}(x, x_1) = T_{kji}(x, x_1), \quad (5.35)$$

and it simply states the symmetry of the Cauchy stress tensor. Lastly we consider the free space Green function defined in (5.27), and the mirroring tensor A defined in (5.25). We obtain the following

$$G_{ij}(x, x_1) = A_{ip}G_{pt}(Ax, Ax_1)A_{tj}. \quad (5.36)$$

$$T_{ijk}(x, x_1) = A_{ip}T_{pqt}(Ax, Ax_1)A_{qk}A_{tj}. \quad (5.37)$$

We introduce a possible free slip kernel as the one associated with two punctual sources located at x_1, Ax_1 , namely

$$G_{ij}^{PS}(x, x_1) = G_{ij}(x, x_1) + G_{ip}(x, Ax_1)A_{pj}, \quad (5.38)$$

that satisfies the following relation

$$\frac{\partial}{\partial x_k} \sigma_x(G_{ij}^{PS}(x, x_1)b_j) = \delta(x - x_1)b_j + \delta(x - Ax_1)A_{pj}b_j. \quad (5.39)$$

Therefore, integrating (5.31) over all the domain Ω , we get

$$\begin{aligned} u_j(x_1)b_j + u_j(Ax_1)A_{jp}b_p &= - \int_{\partial\Omega} f_i(x) [G_{ij}(x, x_1) + G_{ip}(x, Ax_1)A_{pj}] ds(x)b_j \\ &\quad + \int_{\partial\Omega} u_i(x) [T_{ijk}(x, x_1) + T_{ipk}(x, Ax_1)A_{pj}] n_k(x) ds(x). \end{aligned} \quad (5.40)$$

We recover the definitions

$$G_{ij}^{PS} = G_{ij}(x, x_1) + G_{ip}(x, Ax_1)A_{pj}, \quad (5.41a)$$

$$T_{ijk}^{PS} = T_{ijk}(x, x_1) + T_{ipk}(x, Ax_1)A_{pj}, \quad (5.41b)$$

matching (5.28) and (5.30) respectively if we consider properties (5.36), (5.37). At this point we verify that

$$\sigma_{ik} = T_{ijk}^{PS}b_j, \quad (5.42)$$

represents the Cauchy stress tensor associated with a punctual force in x_1 . This is coherent to [93] and [91]. We check that (5.38) satisfies (5.33). We introduce

$$u_i(x) = G_{ij}^{PS}(x, x_1)b_j, \quad (5.43a)$$

$$u'_i(x) = G_{ij}(x, x_2)a_j, \quad (5.43b)$$

taking Figure 5.6 as setting for the problem.

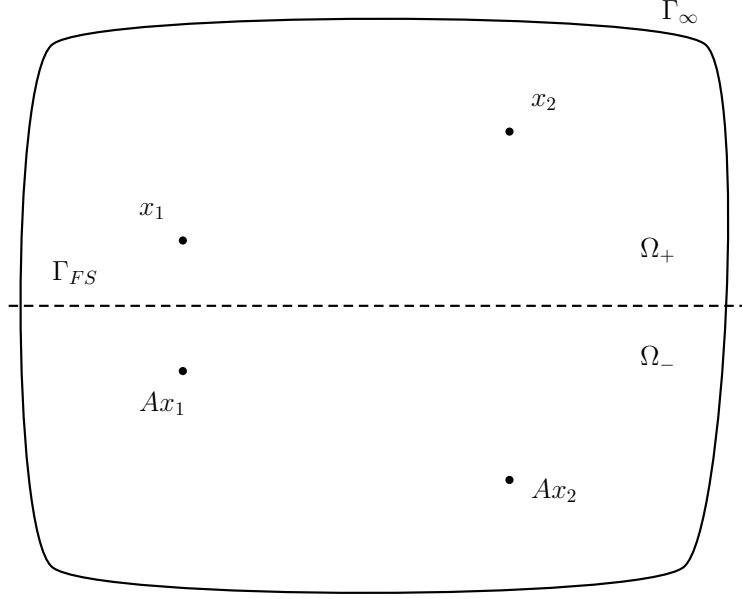


Figure 5.6: The setting used to prove (5.33) using (5.38).

We get

$$\begin{aligned}
 \int_{\Omega_+} \frac{\partial}{\partial x_j} (u'_i \sigma_{ij} - u_i \sigma'_{ij}) &= - \int_{\Gamma_\infty} a_i G_{ij}^{PS}(x, x_1) \sigma_{ik}^{FS'}(x, x_2) - G_{ij}^{FS'}(x, x_2) \sigma_{ik}^{PS}(x, x_1) b_j ds(x) + \\
 &\quad - \int_{\Gamma_{PS}} a_i G_{ij}^{PS}(x, x_1) \sigma_{ik}^{FS'}(x, x_2) - G_{ij}^{FS'}(x, x_2) \sigma_{ik}^{PS}(x, x_1) b_j ds(x) \\
 &= \int_{\Omega_+} a_i G_{ij}^{FS'}(x, x_2) \delta(x - x_1) b_j - b_i G_{ij}^{PS}(x, x_1) \delta(x - x_2) a_j dx \\
 &= a_i G_{ij}^{FS'}(x_1, x_2) b_j - b_i G_{ij}^{PS}(x_2, x_1) a_j.
 \end{aligned} \tag{5.44}$$

G_{ij}^{PS} is composed by two standard free space Green functions vanishing on Γ_∞ . Stresses and velocities related to the reflected Green function are orthogonal on the boundary Γ_{PS} , therefore we have

$$a_i G_{ij}^{FS'}(x_1, x_2) b_j - b_i G_{ij}^{PS}(x_2, x_1) a_j = 0, \tag{5.45}$$

that implies, given the arbitrarily chosen a_j, b_j

$$G_{ij}^{PS}(x_1, x_2) = G_{ji}^{PS}(x_2, x_1). \tag{5.46}$$

We rewrite (5.46) as follows

$$\begin{aligned}
 G_{ij}^{PS}(x_1, x_2) &= G_{ij}(x_1, x_2) + G_{ip}(x_1, Ax_2) A_{pj} \\
 &= G_{ij}(x_2, x_1) + A_{ip} G_{pj}(Ax_1, x_2) \\
 &= G_{ji}(x_2, x_1) + G_{jp}(x_2, Ax_1) A_{pi} \\
 &= G_{ji}^{PS}(x_2, x_1),
 \end{aligned} \tag{5.47}$$

which concludes the proof of (5.33). We rewrite the BIE using the new notation in

(5.48).

$$\begin{aligned} u_j(x_1) + A_{jp}u_p(Ax_1) = & - \int_{\partial\Omega} f_i(x) [G_{ij}(x, x_1) + A_{ip}G_{pj}(x, Ax_1)] ds(x) \\ & + \int_{\partial\Omega} u_i(x) [T_{ijk}(x, x_1) + T_{ipk}(x, Ax_1)A_{pj}] n_k(x) ds(x). \end{aligned} \quad (5.48)$$

In (5.48) the reflection is occurring for the only free index left. If we consider (5.23) instead the reflection is occurring on the other indices. This is coherent since exploiting (5.36) and (5.37) we rewrite (5.48) as

$$\begin{aligned} u_j(x_1) + A_{jp}u_p(Ax_1) = & - \int_{\partial\Omega} f_i(x) [G_{ij}(x, x_1) + G_{ip}(Ax, x_1)A_{pj}] ds(x) \\ & + \int_{\partial\Omega} u_i(x) [T_{ijk}(x, x_1) + A_{ip}T_{pjt}(Ax, x_1)A_{tk}] n_k(x) ds(x), \end{aligned} \quad (5.49)$$

recovering (5.23).

5.3.2 Kernel approximation for two perfect slip interfaces

We approximate two perfect slip interfaces. The analytical solution for such Green kernels is not known. We consider as starting point Section 5.3.1. The flow confined by two perfect slip interfaces present a decay of $1/R$ where R is the distance from the swimmer, this implies that the corresponding kernel should present the same behavior. In [50] the authors suggest that the flow is bi-dimensional, however we know that such a flow maintains precise three-dimensional features and only decays as $1/R$ asymptotically. In particular the kernel T should present a three-dimensional behavior very near the swimmer while showing a bidimensional decay up to some length-scales. As a possible approximation of the two fundamental solution we consider a truncated series of repeated free space solutions G, T . An infinite summation of such kernels does not converge, however, we believe that its truncation could represent a good approximation for the requested Green functions. We introduce

$$G_{ij}^{rep}(x, y) = \sum_{p=-M}^M (G_{ij}(x, y + phe_y)), \quad (5.50a)$$

$$T_{ijk}^{rep}(x, y) = \sum_{p=-M}^M (T_{ijk}(x, y + phe_y)), \quad (5.50b)$$

where h represent the distance between two stacked kernels, we consider the stacking along the y axis. The plane of symmetry in which we analyze the decay corresponds to the mid-plane ($p = 0$). We plot the decay analysis in Figure 5.7. The stacking approximates quite well the desired behavior for the analytical solution of the two perfect slip interfaces, in fact up to $3/4h$ we recover the typical three-dimensional decay $1/r^2$, while up to $5h$ we have the prescribed and expected $1/r$ behavior. We have exploited this approximation in Section 6.5 to simulate the swimming of an eukaryotic organism in an experimental setting consisting of a thin soap film, see [50] for further details on the corresponding experimental setting.

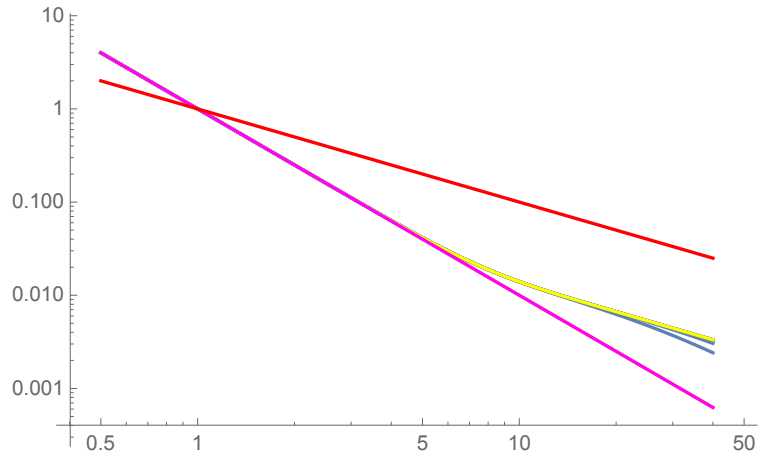


Figure 5.7: Decay for the repeated Green kernel. The kernels are separated by 10 units. In magenta we plot the decay $1/r^2$, in red the decay $1/r$ typical of a 2D environment. In blue we plot two decaying increasing the number of stacked kernel, while in yellow we report the final simulation with 21 stack kernels. We are analyzing the decay on the central plane of symmetry.

CHAPTER 6

Micro-motility analysis using Boundary Element Method

We now move to an application of the Stokes BEM introduced in Chapter 5: the study of swimming strategies of micro-organisms. A Micro-swimmer is a biologically propelled organism that moves through a liquid medium by exploiting periodic shape changes in order to gain a net rigid displacement. Given the size of the problem, the flow around a moving micro-organism can be modeled using the Stokes system. The study of this kind of phenomena has been addressed by several different models [44, 66, 69, 93, 95, 113]. Since swimmers move both in bounded and unbounded domains, BEM (naturally dealing with infinite domains) appears as a convenient choice with respect to standard FEMs. Moreover, the shape changes of the swimmer are often large domain deformations, making BEM a good candidate for the solution of these Fluid Structure Interaction problems, since it requires only a boundary discretization. BEMs have been successfully applied to micro-swimmers over the years [56, 90, 91, 97, 110].

We propose a Boundary Element Method for the Stokes system based on Open-SOURCE High Performance libraries, as the `deal.II` library [6], the Trilinos project [57], and the `deal2lkit` library [102]. We follow the implementation of π -BEM, see Chapter 3, using a hybrid shared-distributed memory parallelization to achieve a higher computational efficiency. We verify the accuracy of the proposed method by validating the code on simple rigidly moving objects and on model swimmers composed by a rigid body and a system of flagella. Such models represent an extensive variety of real organisms as bacteria or flagellated algae. The BEM we propose can be straightforwardly applied to study general swimmers since no assumption on the allowed shape changes has been made in its developing.

While prokaryotic organisms as bacteria exploit a relative rigid rotation of a system of flagella with respect to the body to achieve rigid motion, eukaryotic swimmers,

thanks to their complex flagellar structure, achieve rigid motions exploiting non trivial shape changes. Accurate simulations of the latter kind of organisms are quite rare, examples are [7, 36], given the complexity of retrieving an accurate computational mesh. We coupled an OpenSOURCE three dimensional modeling tool, Blender [18], with Python scripts to obtain a representation of the shape changes in terms of Non Uniform Rational B-Splines (NURBS). Such a representation is extremely general and accurately matches experimental scenarios as the one presented in [50]. This approach is of paramount importance to apply BEM to generic (non-flagellated) swimmers as the ones seen in [7]. We present an application of the complete methodology to the eukaryotic alga described by [50].

The Stokes system is a simplification of the complete mathematical model described by Navier–Stokes equations. Nonetheless, an accurate resolution of the related BIE is far from trivial, and, over the last decades, many simplified methods have been developed to approximate the Stokes flow around micro-organisms. The basic principles of such approximation are the locality of the hydrodynamics interactions and the superimposition principle coming from the linearity of the Stokes system. A notable example is the Resistive Force Theory (RFT) [44, 66, 69, 100], that only considers a relation between local stresses and local velocities. Even from an experimental point of view it is common to analyze separately the different parts of a micro-organism (body and propeller), and then to sum the separate effects [66, 88, 96]. However the ellipticity of the Stokes system requires some non local interactions that may affects the accuracy of the overall simulation. For example in [100] the authors discuss the limitations of RFT when compared with experiments or more accurate numerical methods. Since BEM makes no approximation in the simulation of Stokes flow, it naturally evaluates hydrodynamics interactions not only between different parts of the same body but also between different separated bodies. We consider the work of [96] to discuss and analyze the effect of such interactions on the performance analysis of a flagellated model swimmer.

The chapter is organized as follows. Section 6.1 describes the mathematical modeling of the swimming problem, Section 6.2 details the actual numerical solving procedure. Section 6.3 reports the procedures to obtain a feasible computational mesh. Section 6.4 describes the numerical validation of the proposed methodology on several assessed literature scenarios, and Section 6.5 reports an experimental comparison on the eukaryotic organism studied in [50]. Section 6.6 analyses the interaction between different part of a flagellated bacterium comparing our results with the outstanding analysis of [96].

6.1 Mathematical modeling of micro-swimmers

The interactions between the fluid and the boundary of the swimmer drive the motion of the micro-organism. In particular the only forces acting on the swimmer are of viscous nature, and we assume that no external forces are acting on the organism itself. The shape changes of the boundary cause a stress field on the boundary and since the overall force vanishes a rigid motion must develop to compensate such shape induced stresses. We consider the shape change to be imposed, and we retrieve the rigid motion.

6.1.1 Kinematic model

We follow [28, 73] to derive a mathematical model for the kinematic behavior of a material point of the swimmer, which is modeled as a Lipschitz bounded open set $B_t \in \mathbb{R}^n$ and $n = 2, 3$. We assume that there exists a map $\chi : \bar{B}_0 \subset \mathbb{R}^n \times [0, T] \rightarrow \mathbb{R}^n$ describing the position of a material point of the swimmer. Namely we write

$$x(X, t) = \chi(X, t) = q(t) + R(t)s(X, t), \quad (6.1)$$

where $q(t)$ represents a rigid translation, $R(t)$ is a rotation tensor describing a rotation of the reference frame, and $s(X, t)$ represents a shape change. Clearly $B_t = \chi(B_0, t)$. The swimmer exploits periodic changes of the variable $s(X, t)$ to get a rigid motion described by $q(t), R(t)$.

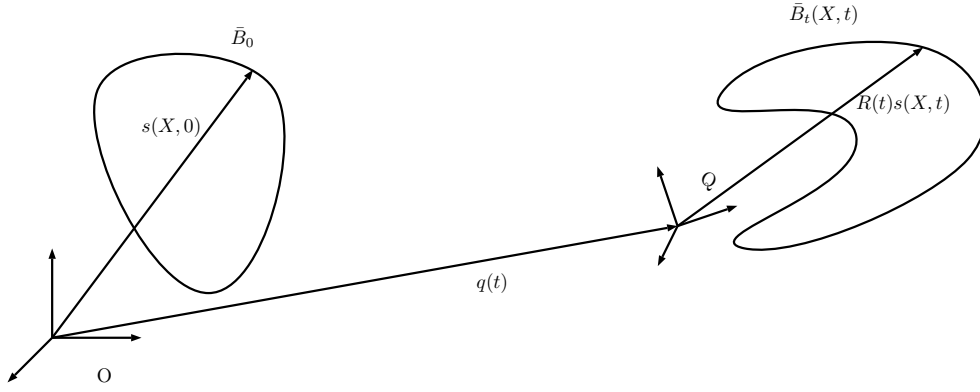


Figure 6.1: Sketch of the kinematic model represented by the map χ . The map is composed by a rigid movement of the reference frame, identified by $q(t), R(t)$ and by some shape changes $s(X, t)$.

Using (6.1) we derive the velocity of any point on the swimmer, and in particular of its boundary Γ , as

$$\begin{aligned} u_{swimmer} = \dot{x} &= \frac{\partial \chi(X, t)}{\partial t} = \frac{dq}{dt}(t) + \frac{dR(t)}{dt}s(X, t) + R(t)\frac{\partial s(X, t)}{\partial t} = \\ &= \dot{q}(t) + \dot{R}(t)R^T(t)R(t)s(X, t) + R(t)\dot{s}(X, t), \\ &= \dot{q}(t) + \omega(t) \wedge (R(t)s(X, t)) + R(t)\dot{s}(X, t). \end{aligned} \quad (6.2)$$

We assume $s(X, t)$ to be known. The unknowns are $q(t)$ and $R(t)$ and their derivatives $\dot{q}(t), \omega(t)$, we note that $R(t)s(X, t)$ expresses the current position of the point in a reference frame attached to the body but aligned to the inertial frame O . We group the summands of (6.2) in two parts representing velocities due to rigid movements and shape changes respectively, namely

$$u_{swimmer} = u_{rigid}(X, t) + v(X, t) \quad (6.3)$$

where

$$v(X, t) = R(t)\dot{s}(X, t), \quad (6.4)$$

we highlight that $v(X, t)$ is known only if the actual configuration of the swimmer is known. To express $u_{rigid}(X, t)$ we need a set of basis functions to express the rigid

velocities of the swimmer, namely we need basis functions both for the linear and for the angular velocity. We write

$$\dot{q}(t) = \sum_{i=1}^d e_i \dot{q}_i(t) = \sum_{i=1}^d p_i^{\dot{q}} \dot{q}_i(t), \quad (6.5a)$$

$$\omega(t) \wedge R(t)s(X, t) = \sum_{i=1}^{N_\omega} \omega_i(t) e_i \wedge R(t)s(X, t) = \sum_{i=1}^{N_\omega} p_i^\omega(X, t) \omega_i(t), \quad (6.5b)$$

we rewrite (6.2), grouping together all the rigid modes, as

$$\begin{aligned} u_{swimmer} &= \dot{q}(t) + \omega(t) \wedge R(t)s(X, t) + v(X, t) \\ &= \sum_{i=1}^{N_{rigid}} p_i(X, t) \dot{p}_i(t) + v(X, t) \\ &= P(X, t) \dot{p}(t) + v(X, t), \end{aligned} \quad (6.6)$$

where $N_{rigid} = d + N_\omega$ (3 if $d = 2$, 6 if $d = 3$), and $\dot{p}_i(t) = \dot{q}_i(t)$ if $i < d$ and $\dot{p}_i(t) = \omega_{i-d}(t)$ otherwise. We remark that we use the $\dot{\cdot}$ notation in the vector $\dot{p}(t)$ even if it does not represent strictly a classical time derivative, since it consists of both rigid linear and angular velocities (which are not directly the derivatives of $R(t)$).

6.1.2 Fluid model

We are interested in swimmers moving in water, so we model the fluid as an incompressible Newtonian material. We focus our attention on the system of equations formed by the mass conservation and the linear momentum balance. We identify the micro swimmer as the closure of an open Lipschitz bounded set inside the fluid domain $B_t \in \mathbb{R}^d$ with $d = 2, 3$. We call the boundary of the swimmer $\Gamma = \partial B_t$. The swimming mechanism is based on a periodic shape change that we interpret as a periodic change of Γ . The domain in which we are interested to solve the fluid equations is Ω as shown in Figure 5.1. We consider the balance laws for an incompressible Newtonian, we indicate as σ the stress tensor in the fluid, namely

$$\sigma = \sigma(u, p) = 2\mu \left(\frac{1}{2} \nabla u + \frac{1}{2} \nabla u^T \right) - pI, \quad (6.7)$$

where we indicate with u the velocity of the fluid and with p the associated pressure. Using (6.7), we write the mass and linear momentum balance laws as

$$\nabla \cdot u = 0 \text{ in } \mathbb{R}^d, \quad (6.8a)$$

$$\rho \left[\frac{\partial u}{\partial t} + (\nabla \cdot u)u \right] = \nabla p + \mu \Delta u \text{ in } \mathbb{R}^d, \quad (6.8b)$$

we note that since the fluid is incompressible, the pressure is no longer a thermodynamic variable but it is a Lagrange multiplier needed to impose equation (6.8a).

We proceed with a classical adimensionalization of the equations of motion (6.8). We consider some reference quantities: L as reference length, U as reference velocity,

6.1. Mathematical modeling of micro-swimmers

ρ as reference density and μ as reference viscosity. Since we consider micro-swimmers moving without constraints and in calm water we assume L as the main dimension of the cell and U as the swimming velocity. We take L/U as characteristic time and $\mu U/L$ as characteristic pressure. We get

$$\nabla \cdot \hat{u} = 0 \text{ in } \Omega, \quad (6.9a)$$

$$\frac{\partial \hat{u}}{\partial \hat{t}} + (\nabla \cdot \hat{u})\hat{u} = \frac{1}{Re} (\nabla \hat{p} + \Delta \hat{u}) \text{ in } \Omega. \quad (6.9b)$$

where $\hat{\cdot}$ indicates the adimensional variable, *i.e.* $u = \hat{u}U$ and Re is the so-called Reynolds number,

$$Re = \frac{\rho UL}{\mu}. \quad (6.10)$$

A back of the envelope calculation of Re can be obtained by considering that the body dimension is of the order of $10^{-5}m$, the velocity is also of the order of $10^{-5} \frac{m}{s}$. Typically μ is of the order of $10^{-3} \frac{Pa}{s}$ while ρ is of the order of $10^3 \frac{kg}{m^3}$. So we get Re of the order of 10^{-4} , and the inertial terms are negligible in comparison to the viscous components of the momentum balance. These considerations lead us to the Stokes system,

$$\nabla \cdot u = 0 \text{ in } \Omega, \quad (6.11a)$$

$$\Delta u + \nabla p = 0 \text{ in } \Omega, \quad (6.11b)$$

where we dropped the $\hat{\cdot}$ notation for the sake of simplicity. As boundary condition we assume standard adhesion on the swimmer boundary

$$u = u_{swimmer} \text{ on } \partial B_t, \quad (6.12)$$

while the boundary conditions on Γ_w depend on the specific interface we are considering. This system is well known, it has the typical structure of saddle point problems and admits a unique solution $u \in (H^1(\Omega))^d, p \in L_2(\Omega)$. We can uniquely relate the stresses $f = \sigma(u, p)n$ on the boundary Γ to the Dirichlet datum $u_{swimmer}$, to do so we use the BEM introduced in Chapter 5.1, and we define the so-called Dirichlet to Neumann Map

$$T : (H^{\frac{1}{2}}(\Gamma))^d \rightarrow (H^{-\frac{1}{2}}(\Gamma))^d, \quad (6.13)$$

using (5.10) we express T as

$$T : [V]^{-1}[K]. \quad (6.14)$$

We apply (6.14) to rewrite the stresses as

$$f = \sigma(u, p)n = Tu_{swimmer}. \quad (6.15)$$

6.1.3 Swimmer model

We repeat the dimensional analysis on the swimmer itself and we recover that its inertial forces are negligible. This implies that at any time the overall force and torque on the swimmer must be null since we are neglecting inertia. The following system of equation is satisfied for any micro-swimmer we are modelling,

$$\int_{\Gamma} f(x) d\gamma(x) + \int_A f_b(x) dx = 0, \quad (6.16a)$$

$$\int_{\Gamma} f(x) \wedge (x - x_0) + m_s d\gamma(x) + \int_A f_b(x) \wedge (x - x_0) + m_b dx = 0. \quad (6.16b)$$

Where we have called x_0 a generic pole for the swimmer the and with x a generic application point of the force. We have called f the stresses acting on the surface of the body and f_b the distributed forces acting on the volume of the body, while m_s and m_b represent the torque sources on the surface and in the body respectively. From classical continuum mechanics we express the force acting on the surface of the swimmer as the action of the Cauchy stress tensor, namely

$$f = \sigma(u, p)n, \quad (6.17)$$

where n indicates the normal vector to the surface, pointing outwards the body and inwards the fluid domain. We follow the hypothesis of no volume forces or torques so $f_b = 0$ and $m_b = 0$. On the surface of the body the forces must be generated by the fluid and by classical continuum mechanics [51] we have $m_s = 0$. We use (6.15) to rewrite (6.16) as

$$\int_{\Gamma} Tu \, d\gamma = 0, \quad (6.18a)$$

$$\int_{\Gamma} Tu \wedge (x - x_0) \, d\gamma = 0. \quad (6.18b)$$

The pole x_0 in equation (6.18b) is arbitrary since no external forces are acting on the swimmer, except viscous ones. We apply (6.2) to rewrite (6.18) as

$$\int_{\Gamma} T(\dot{q}(t) + \omega(t) \wedge (R(t)s(X, t)) + R(t)\dot{s}(X, t)) \, d\gamma = 0, \quad (6.19a)$$

$$\int_{\Gamma} T(\dot{q}(t) + \omega(t) \wedge (R(t)s(X, t)) + R(t)\dot{s}(X, t)) \wedge (x - x_0) \, d\gamma = 0. \quad (6.19b)$$

We use 6.6 to rewrite (6.19) as

$$\int_{\Gamma} T(P(X, t)\dot{p}(t) + v(X, t)) \, d\gamma = 0, \quad (6.20a)$$

$$\int_{\Gamma} T(P(X, t)\dot{p}(t) + v(X, t)) \wedge (x - x_0) \, d\gamma = 0, \quad (6.20b)$$

using the definition of rigid modes introduced in (6.6) we can rewrite (6.20) in a more compact form as

$$\int_{\Gamma} P^T T(P(X, t)\dot{p}(t) + v(X, t)) = 0. \quad (6.21)$$

We reorder (6.21) to obtain

$$\mathcal{R}(X, t)\dot{p}(t) + \theta(X, t)v(X, t), \quad (6.22)$$

where

$$\mathcal{R} = \mathbb{R}^{N_{rigid}} \rightarrow \mathbb{R}^{N_{rigid}}, \quad (6.23a)$$

$$\mathcal{R}_{ij} = \int_{\Gamma} p_i(X, t) T p_j(X, t) d\gamma, \quad (6.23b)$$

6.2. Numerical resolution of the swimming problem

and

$$\theta : H^{\frac{1}{2}}(\Gamma) \rightarrow \mathbb{R}^{N_{rigid}}, \quad (6.24a)$$

$$(\theta u)_i : \int_{\Gamma} p_i(X, t) T u(x) d\gamma, \quad (6.24b)$$

where $\mathcal{R}(t)$ is the symmetric positive definite Grand Resistance Matrix describing the forces and torques due to the N_{rigid} rigid modes, and $\theta(t)$ is an operator expressing the forces and torques induced by a prescribed velocity field on Γ . We remark that $\mathcal{R} = \theta P$, and that from (6.5) we see that the rigid modes depend on the current body orientation R and the current shape R_s so both \mathcal{R} and θ are specific for the swimmer configuration at time t . We remind that the Grand Resistance Matrix is finite dimensional ($N_{rigid} \times N_{rigid}$) and can be inverted (symmetric and positive definite) to obtain the rigid velocities as functions of the velocity shape change, namely we write

$$\dot{p}(t) = -\mathcal{R}^{-1}(X, t) \theta(X, t) v(X, t). \quad (6.25)$$

Once we have obtained the rigid velocities \dot{p} we integrate them to obtain the rigid displacement. We use separate strategies to integrate the linear parts. In particular we use quaternions to integrate the angular velocity and to parametrize the rotation tensor $R(t)$.

6.2 Numerical resolution of the swimming problem

We combine the discretization of the Stokes Boundary Integral Equation described in Section 5.2 with the hypotheses of Section 6.1 to solve the swimming problem. In particular, we make the following assumptions: the reference shape of the swimmer is given by a surface description of its geometry, and we discretize it using the assumptions expressed in Chapter 5 (see page 90, eq (5.14)).

The instantaneous shape of the swimmer $s(X, t)$ is a (given) data obtained through a (known) deformation field η , such that

$$s(X, t) = s_0(X) + \eta(X, t). \quad (6.26)$$

We exploit iso-parametric discretizations of Γ_0 , and use a standard interpolation operator on Γ_0 to interpolate the data of the problem:

$$\begin{aligned} \Pi : H^{\frac{1}{2}}(\Gamma) &\rightarrow V_h, \\ v(X, t) &\rightarrow \sum_{i=1}^{dN_V} v_i(t) \psi_i(X), \quad X \in \Gamma_0 = \partial B_0. \end{aligned} \quad (6.27)$$

Using Π , s_0 , and η , s can be interpolated on the finite dimensional space, giving

$$s_{0,h}(X) = \sum_{i=1}^{dN_V} \psi_i(X) s_{0,i} = \Pi s_0(X), \quad (6.28)$$

and

$$\eta_h(X, t) = \sum_{i=1}^{dN_V} \psi_i(X) \eta_i(t) = \Pi \eta(X, t), \quad (6.29)$$

where we notice that the dependency on time remains in the coefficients of the deformation field η_h . The actual configuration reads

$$s_h(X, t) = s_{0,h}(X) + \eta_h(X, t), \quad (6.30)$$

we remind that $s_h(X, t)$ is the datum for the considered problems. We define the finite dimensional approximation of the shape velocity v as

$$v_h(X, t) = R(t)\dot{s}_h(X, t) = R(t)\dot{\eta}_h(X, t), \quad (6.31)$$

and we use Π to write the finite dimensional approximation of the N_{rigid} rigid modes $p_i(X, t)$ as,

$$p_{i,h}(X, t) = \Pi p_i(X, t), \quad (6.32)$$

consequently, with a little abuse of notation we define the matrix P_{ij} so that

$$p_{i,h}(X, t) = \sum_{j=1}^{dN_V} P_{ij} \psi_j(X), \quad (6.33)$$

and we drop the $(\dots)_h$ notation for the sake of simplicity. We write the finite dimensional solving system introducing the system matrix

$$A = \begin{bmatrix} V & -KP \\ P^T M & 0 \end{bmatrix} \quad (6.34)$$

where M represents the Mass matrix that performs the surface integration as depicted in (6.16). We only need to define the right hand side starting from the known shape velocities (6.35)

$$b = \begin{bmatrix} Kv \\ 0 \end{bmatrix}. \quad (6.35)$$

The complete monolithic system is presented in (6.36)

$$\begin{bmatrix} V & -KP \\ P^T M & 0 \end{bmatrix} \begin{bmatrix} f \\ \dot{p} \end{bmatrix} = \begin{bmatrix} Kv \\ 0 \end{bmatrix}. \quad (6.36)$$

The system (6.36) is equivalent to (6.25), in fact the first equation reads

$$f = V^{-1}K(P\dot{p} + v) = Tu, \quad (6.37)$$

which is the finite dimensional equivalent of (6.15). The second line 6.36 reads

$$P^T M T u = 0, \quad (6.38)$$

and it expresses the linear and angular momentum balance laws in the finite dimensional setting. System (6.36) is solved using a parallel direct linear solver. However if the number of unknowns increases, such a strategy is very demanding from a computational point of view. We are currently studying more efficient precondition strategies for the system matrix A , in order to use preconditioned iterative Krylov solvers. At the moment we assemble either a Jacobi or an Algebraic MultiGrid preconditioner starting from A_{prec} defined as

$$A_{prec} = \begin{bmatrix} V & -KP \\ P^T M & I \end{bmatrix}. \quad (6.39)$$

6.3. Geometrical reconstruction of a micro-swimmer

The use of an iterative solver reduces the computational requirements but it is extremely sensible on the choice of an effective preconditioner. In particular we are currently studying the effectiveness of the preconditioner in presence of physical interfaces as no-slip or perfect-slip walls. We subdivide the domain in different subset, namely

$$\Gamma_h = \Gamma_h^{body} \cup \Gamma_h^{no-slip} \cup \Gamma_h^{perfect-slip} \cup \Gamma_h^{dirichlet} \cup \Gamma_h^{neumann}, \quad (6.40)$$

where no intersection is possible between different subsets. Depending on the position of the support point we need to consider different entries for the monolithic system matrix A .

- $A(i, j) = V(i, j)$ if $x_j \in \Gamma_h^{body}$, we impose the shape velocities and we need to retrieve the unknowns stresses.
- $A(i, j) = V(i, j)$ if $x_j \in \Gamma_h^{no-slip}$, we impose null shape velocities and we need to retrieve the unknowns stresses on a meaningful interface.
- $A(i, j) = -K(i, j)$ if $x_j \in \Gamma_h^{perfect-slip}$ and $n_j = 0$, we are in the tangent direction on a perfect slip interface thus we impose null stresses to retrieve the velocity on such an interface .
- $A(i, j) = V(i, j)$ if $x_j \in \Gamma_h^{perfect-slip}$ and $n_j = 1$, we consider the normal direction to a perfect slip surface, therefore we impose null velocity to recover the stress in such direction.
- $A(i, j) = V(i, j)$ if $x_j \in \Gamma_h^{dirichlet}$, we impose homogeneous Dirichlet boundary condition on an interface lying far away from the body, this is allowed to better approximate infinity radiation conditions.
- $A(i, j) = -K(i, j)$ if $x_j \in \Gamma_h^{neumann}$ we impose homogeneous Neumann boundary condition on an interface lying far away from the body, this is allowed to better approximate infinity radiation conditions.

Once we have solved (6.36) we integrate the rigid velocities to obtain the rigid displacement. We use quaternions to parametrize the rotation matrix $R(t)$ and we exploit the computed ω to update the quaternion.

6.3 Geometrical reconstruction of a micro-swimmer

An essential key step to model micro-swimmer is the reconstruction of its geometry. We describe the actual procedure to analyze a series of experimental images and reconstruct a suitable computational mesh. To achieve higher level of freedom in 3D modeling use the 3D OpenSource creation suite Blender, see [18], which is highly compatible with existing HPC libraries and is highly tunable using standard Python coding. We consider a flagellated micro-organism called *Chlamydomonas Reinhardtii*. We use images from Professor Jeffrey S. Guasto, see [50] or <http://sites.tufts.edu/guastolab/movies/>. The key steps of our procedure are

- flagellum tracking;
- rigid motion tracking;

- flagellum reparametrization using NURBS curves;
- 3D Modeller NURBS importer;
- grid Creation.

Flagellum tracking We track the flagella using an open source software for image analysis, ImageJ [104]. Figure 6.2 reports the tracking of a flagellum at a given frame. If

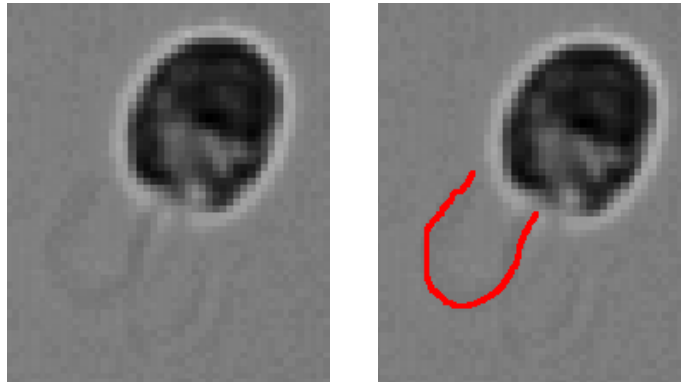


Figure 6.2: *On the left one of the original frames obtained by Professor Jeffrey S. Guasto, see [50]. On the right the same track with the right flagellum tracked.*

the resolution of the image is not enough for an automatic tracking we manually select the flagellum frame by frame.

We express the tracked results in a standard format (`.txt`) and modify them in a scientific environment (Python).

Rigid motion tracking If the set of images presents a rigid motion we need to track it and subtract it from the flagella movements we have previously obtained. The tracking is also used as an experimental validation of our Stokes simulator. The rigid tracking simply consists in choosing a reference section of the image and then track its movements using a correlation scheme.

To set up the marker that will be tracked, we choose a point in the image inside the non deforming parts of the organism. Through a correlation we retrieve the tracking over all the images, which is then exported to a text file. We report the marking for the rigid tracking in Figure 6.3.

Flagellum reparametrization using NURBS curves The raw flagella files are postprocessed to set up the computational mesh. We import the raw files in Python to generate a NURBS approximation and then we import them in the modeling tool via its Python interface. In Figure 6.4 we see the raw flagellum data. Such a format is not suited for the modeling of the computational mesh but can easily be processed, by re-parameterizing it using NURBS, through a Least Square approximation to create the desired curve, where we additionally impose an inextensibility constraint. Usually organic flagella are inextensible, thus we require all the curves to have the same length. Since the type junction between body and flagella is not completely clear from the microscopic images, we have required an horizontal tangent to the curve at the root point, see Figure

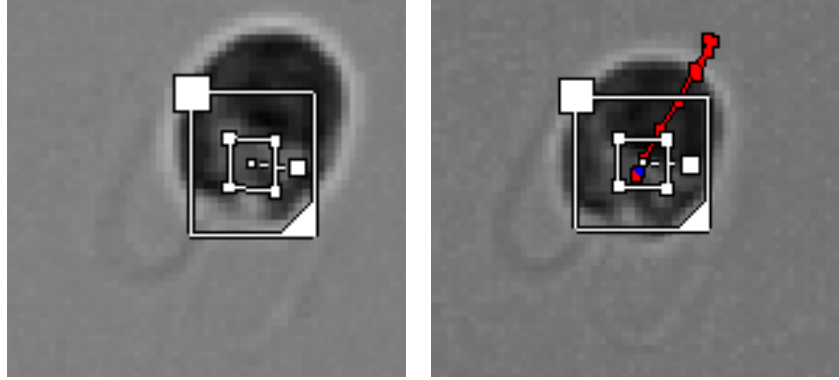


Figure 6.3: On the left we report the setting for the tracking on the original images obtained by Professor Jeffrey S. Guasto, see [50]. The inner square represents the piece of image that we will search. The outer square represents the searching area in which we will try to find the marker. On the right the red path represents the track of the marker.

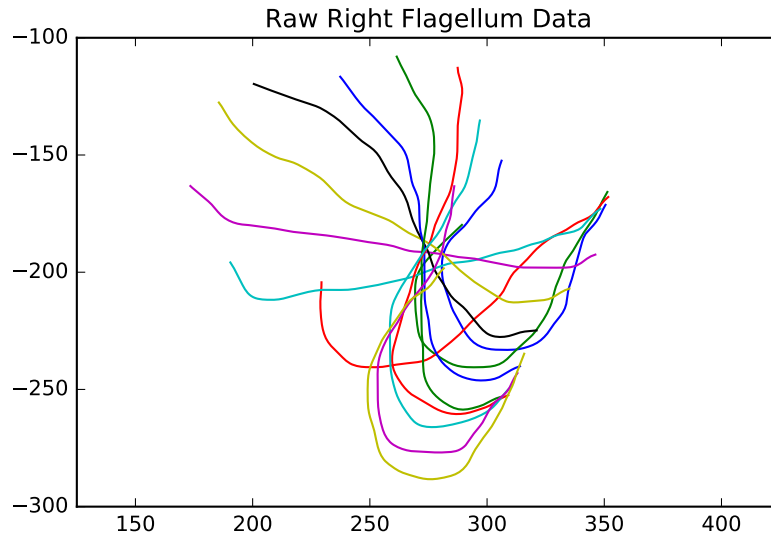


Figure 6.4: The raw tracks of the right flagellum. These curves were exported after a manual tracking of the flagellum.

6.6 . This represents a natural clamp constraint at the end of the flagellum. We see that the resulting curve is very close the original frame captures.

3D modeller NURBS importer We import the control points of the curve in the modeler, and we use them to animate the curve in the modeler. Since the number of frames we have tracked is not enough for a reliable simulation, we import the control points at selected frame distance and then we let the modeler interpolate linearly between the given frames. In Figure 6.7 we see a snapshot of the animated curve.

Grid creation In the example we consider, the swimmer is composed by two moving flagella and a rigid body. A reference undeformed grid is created all the symmetries

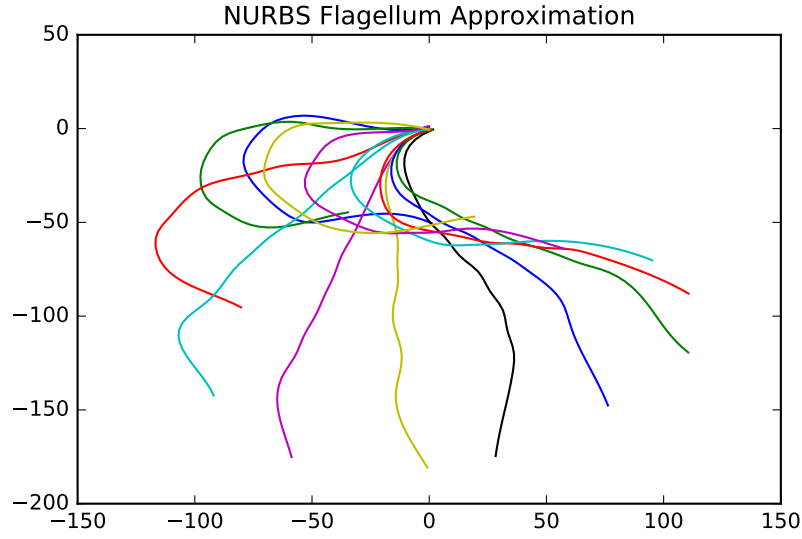


Figure 6.5: *The Least Square approximations of the flagellum data. All the curves have the same length.*

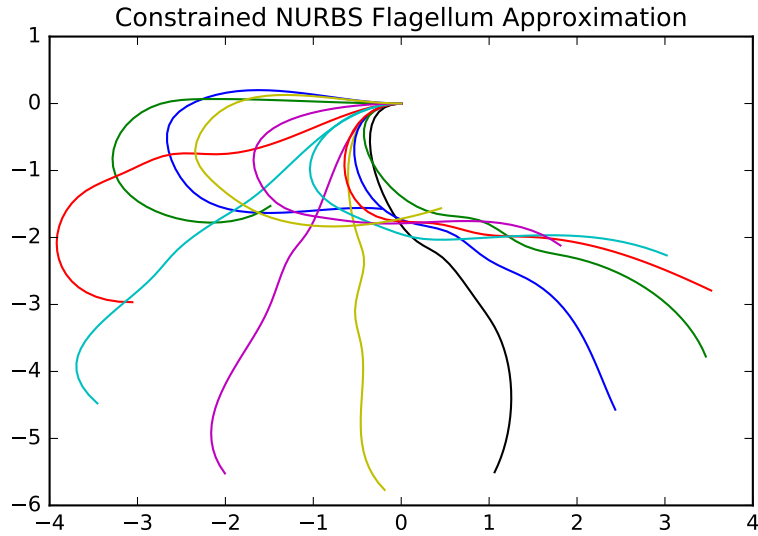


Figure 6.6: *Final NURBS approximation obtained using a constrained reparametrization.*

of the swimmer. The flagella part of the grid is deformed by imposing it to follow the curve created in the reparametrization step. In Figure 6.8 we see the undeformed computational grid for a two-dimensional simulation. In Figure 6.9 we see instead the computational grid for a three-dimensional simulation. We use a spherical body for the microswimmer. In Figure 6.10 we see the two complete computational grids. The complete animation can be seen at https://www.youtube.com/watch?v=SX_VCyirTEk&feature=youtu.be. The final computational grids are exported to format compatible with the `deal.II` library. In particular we use Assimp, a

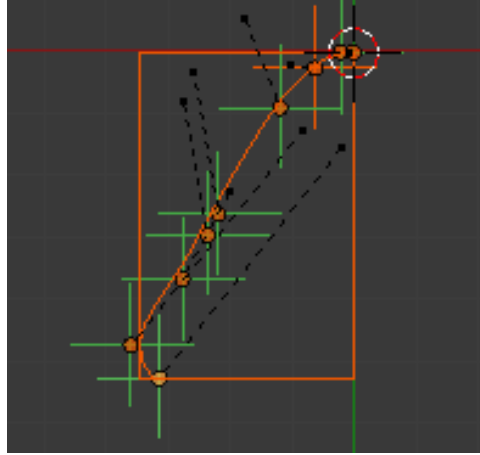


Figure 6.7: The animated NURBS in the three-dimensional modeling software. The green crosses represent the knots of the NURBS at a selected time frame

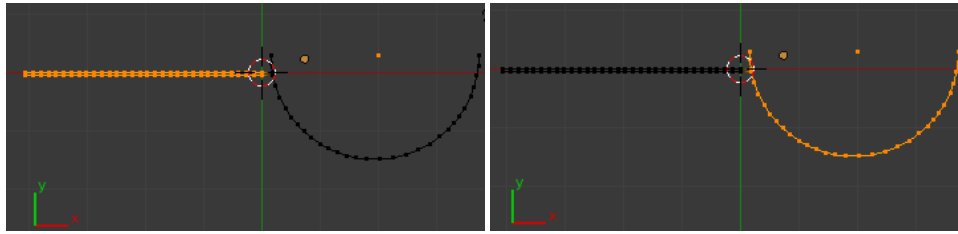


Figure 6.8: The two-dimensional computational grid. On the left the part we use to simulate the flagella, on the right the part we use to simulate the body.

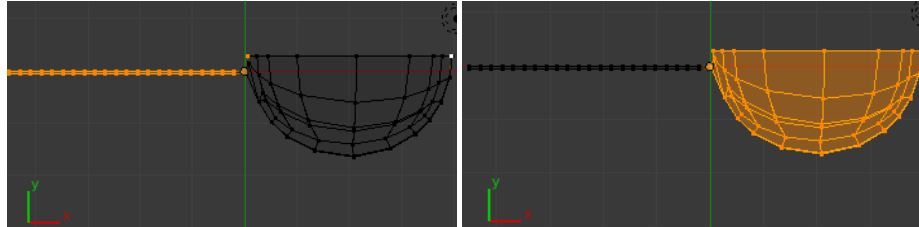


Figure 6.9: The three-dimensional computational grid. On the left the part we use to simulate the flagella, on the right the part we use to simulate the body.

portable Open Source library able to import various well-known 3D model formats in a uniform manner, the Assimp Interface is currently provided by the `deal21kit` library, see [103].

6.4 Numerical validation

We verify the accuracy of the developed BEM method on Benchmark applications. Simple rigidly moving objects and composite swimmers are the benchmarks we use to validate the proposed method. In particular we consider spheres, spheroids and spirals as rigidly moving objects. The composite bacterium model consists in a rigid head and a single rotating flagellum (modeled as a circular helix). Section 6.4.1 assesses the accuracy of the method on different bodies (spheres or ellipsoids) and flagella (helices)

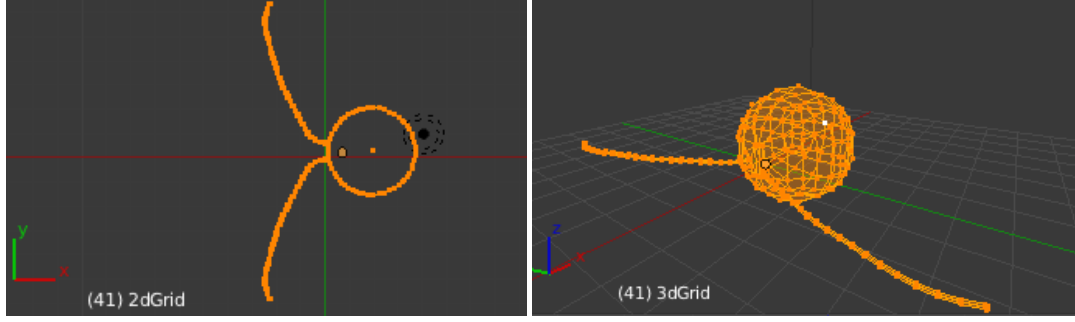


Figure 6.10: The deformed computational meshes. On the left we see the two-dimensional grid and on the right the three-dimensional grid.

by computing their Resistance matrices. Section 6.4.2 analyzes the drag experienced by a sphere falling toward a physical interface. In Section 6.4.3 we study a test case concerning the interactions between two rigid bodies considering two rigid spheres dragged along their centerline. Finally in Section 6.4.4 we address the study of a complete bacterium both in free space and near interfaces (both no-slip and perfect slip ones).

6.4.1 Calculation of resistance matrices

From Stokes linearity the forces acting on a rigid body linearly depend on its velocities, through the resistance matrix \mathcal{R} introduced in (6.23), namely

$$\mathcal{F} = \mathcal{R}\mathcal{U}, \quad (6.41)$$

where \mathcal{F} represents the forces on a rigid body and torques acting on the body while \mathcal{U} its linear and angular velocities. Following [66] \mathcal{R} denotes the entries of the viscous resistance matrices for the considered geometries. For all the considered resistance matrix entries we use the notation $\mathcal{R}_{ij}^{\alpha\beta}$, where the superscript is either FU depicting the i th force component induced by the j th linear velocity, $F\Omega$ describing the i th force component generated by the j th angular velocity, LU representing the i th torque component induced by the j th linear velocity, $L\Omega$ depicting the i th torque component generated by the j th angular velocity.

Sphere and spheroid

If the body is axialsymmetric the resistance matrix \mathcal{R} is diagonal. In particular from both the sphere and the spheroid have a diagonal resistance matrix, as the one shown in (6.42).

$$\mathcal{R}_{sphere}^{Theory} = \begin{bmatrix} \mathcal{R}_{xx}^{FU} & & & & & \\ & \mathcal{R}_{yy}^{FU} & & & & \\ & & \mathcal{R}_{zz}^{FU} & & & \\ & & & \mathcal{R}_{xx}^{L\omega} & & \\ & & & & \mathcal{R}_{yy}^{L\omega} & \\ & & & & & \mathcal{R}_{zz}^{L\omega} \end{bmatrix} \quad (6.42)$$

We expect the force coefficients to be equal to $F_{sphere} = 6\pi\mu RV$, and the torque ones to be equal to $T_{sphere} = 8\pi\mu R^3\omega$, where R is the radius of the sphere and μ the viscosity of the fluid. We consider $R = 1$ and $\mu = 1$ we expect $F_{sphere} = 18.85$ and $T_{sphere} = 25.13$. We retrieve $F_{sphere} = 18.80$ and $T_{sphere} = 24.492$

We compute the resistance matrix for a prolate spheroid. The major axis is aligned with the x axis and is long $b_1 = 2$, while the others are long $b_2 = 1$. While the resistance matrix is still diagonal we expect the drag coefficients to be different as shown in (6.43)

$$\begin{aligned}\mathcal{R}_{xx}^{FU} &= 6\pi\mu b_1 \frac{\frac{16}{3}e^3}{2e + (3e^2 - 1.) \log(\frac{1+e}{1-e})}, \\ \mathcal{R}_{yy}^{FU} = \mathcal{R}_{zz}^{FU} &= 6\pi\mu b_1 \frac{\frac{8}{3}e^3}{-2e + (3e^2) \log(\frac{1+e}{1-e})}.\end{aligned}\tag{6.43}$$

Using (6.43) and considering the present test case we obtain $\mathcal{R}_{xx}^{FU} = 22.51$, $\mathcal{R}_{yy}^{FU} = \mathcal{R}_{zz}^{FU} = 25.99$, and we recover the same accuracy obtained for the sphere.

Spiral

We model the flagellum as a perfect helix with amplitude b , wavelength λ , number of turns N_λ , and flagellar radius r . We sketch the geometry in Figure 6.11. We measure the mean values for the coefficients \mathcal{R} during a stroke, intended as a complete rotation of $\phi = 2\pi$ along the longitudinal axis. Given the symmetries associated with a complete

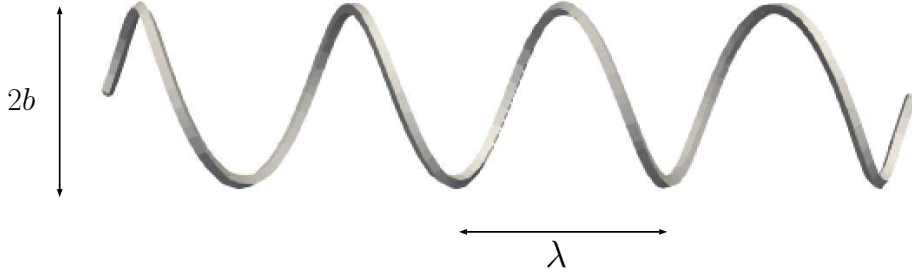


Figure 6.11: Geometry for the flagellum as reported in [100]. We consider the flagellum as a cylinder with radius r mapped on a spiral of wavelength λ and amplitude b . In this example we consider $N_\lambda = 4$.

rotation in free space we expect some coefficients to vanish. From [66] we expect the pattern shown in (6.44)

$$\mathcal{R}_{spiral}^{Theory} = \begin{bmatrix} \mathcal{R}_{xx}^{FU} & & & \mathcal{R}_{xx}^{F\omega} & & \\ & \mathcal{R}_{yy}^{FU} & & \mathcal{R}_{yy}^{F\omega} & \mathcal{R}_{yz}^{F\omega} & \\ & & \mathcal{R}_{zz}^{FU} & \mathcal{R}_{zy}^{F\omega} & \mathcal{R}_{zz}^{F\omega} & \\ \mathcal{R}_{xx}^{LU} & & & \mathcal{R}_{xx}^{L\omega} & & \\ & \mathcal{R}_{yy}^{LU} & \mathcal{R}_{yz}^{LU} & \mathcal{R}_{yy}^{L\omega} & & \\ & \mathcal{R}_{zy}^{LU} & \mathcal{R}_{zz}^{LU} & & \mathcal{R}_{zz}^{L\omega} & \end{bmatrix}.\tag{6.44}$$

We report the matrix obtained with our method in (6.45). As test case we have considered $b = 0.2$, $\lambda = 2.5$, $N_\lambda = 3$, and $r = 0.05$.

$$\mathcal{R}_{spiral}^{FreeSpace} = \begin{bmatrix} 12.904 & & & -0.562 & & \\ & 19.046 & & & 0.201 & 90.458 \\ & & 19.046 & & -90.460 & 0.204 \\ & & & 1.581 & & \\ & 0.201 & -90.460 & & 552.180 & \\ 90.458 & 0.203 & & & & 552.157 \end{bmatrix} \quad (6.45)$$

We recover the pattern expected, this is coherent with [66].

The coefficients appearing in (6.44) highly depend on the spiral geometrical parameters, for this reason we follow [100] to perform an analysis of the forces acting on a spiral flagellum. In [100] Rodenborn *et al.* show an extremely meaningful comparison between different methods, from RFT to Regularised Stokeslet method (very similar to the BEM developed here) on the spiral geometry. We consider $r = b/16$, $\lambda = 2.42b$, and we let N_λ vary between 1 and 14. The motility of a bacterium in free space is characterized by three main coefficients: $\mathcal{R}_{xx}^{F\Omega}$, $\mathcal{R}_{xx}^{L\Omega}$ and \mathcal{R}_{xx}^{FU} . $\mathcal{R}_{xx}^{F\Omega}$ describes the propulsive force F induced by the spiral rotation, $\mathcal{R}_{xx}^{L\Omega}$ depicts the reacting torque T induced by the flagellum rotation and \mathcal{R}_{xx}^{FU} defines the drag D induced by a translation with unit velocity. We report our comparisons in Figures 6.12a, 6.12b and 6.12c. From Figure 6.12 we see a very good agreement between the present method and the expected experimental and numerical results by Rodenborn *et al.*, see [100].

6.4.2 Towed sphere near a wall

We analyze the motion of a sphere towed perpendicularly to a physical interface. We consider the reference theoretical solution reported by Happel and Brenner in [53]. ρ describes the distance between the wall and the center of the sphere, and we impose a unit velocity on the sphere. We compute the corresponding drag acting on the sphere and we normalise it by the reference free space drag for a sphere. In Figure 6.13a and 6.13b we compare the results considering a no slip and a perfect slip interface respectively. We plot in blue the analysis of the present method, while the green squares represent the expected solution. In both cases we see a very good agreement between the results obtained using our BEM and the theoretical solutions. We remark that at the lowest distance considered $\rho = 1.12R$ the error increases due to the geometrical reconstruction of the discrete mesh. This is expected since the distance between sphere and wall is smaller than the mesh size on both the sphere and the wall.

6.4.3 Two spheres analysis

We consider the hydrodynamic interactions between two rigid bodies in a Stokes flow. We analyze two equal spheres of radius R that are separated along their centerline by a distance ρ . We compute the drag induced on the system by a velocity \bar{U} parallel to the centerline and the velocity due to an imposed force \bar{F} directed again along the centerline. We use the exact solution reported by Happel and Brenner in [53] as benchmarks. We plot the results in Figure 6.14, on the left we present the comparison between the

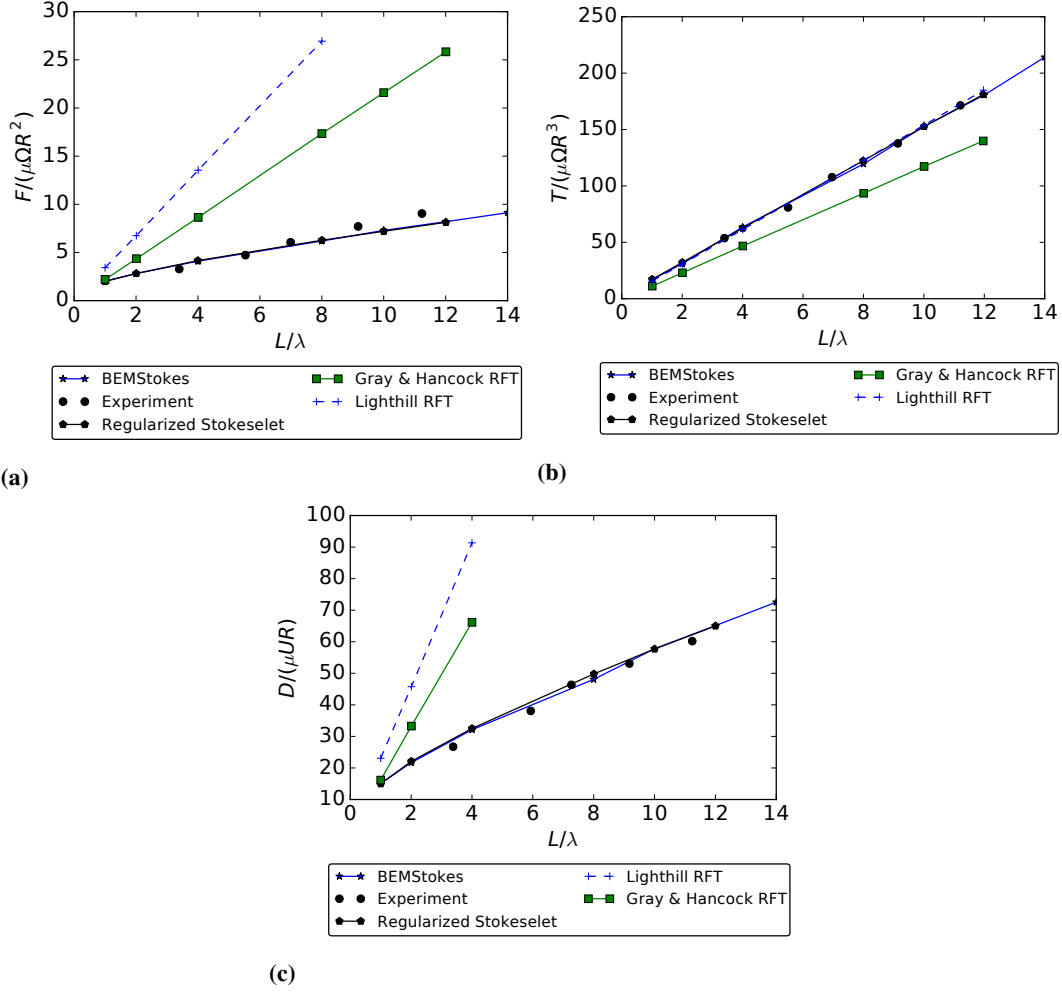


Figure 6.12: Comparison of the adimensional resistance coefficient $\mathcal{R}_{xx}^{F\omega}, \mathcal{R}_{xx}^{L\omega}, \mathcal{R}_{xx}^{FU}$. Figure 6.12a represents the comparison for the force coefficient $\mathcal{R}_{xx}^{F\omega} = F$, Figure 6.12b describes the comparison for the torque coefficient $\mathcal{R}_{xx}^{L\omega} = T$ and Figure 6.12c depicts the comparison for the drag coefficient $\mathcal{R}_{xx}^{FU} = D$. The black dots represent the experimental result, the black line with pentagons depicts the Regularized Stokeslet method, while the green plot and the dashed one describe RFT developed by Gray & Hancock and Lighthill respectively. The continue starred blue line shows the results of the present method.

force acting on each sphere and the drag of an isolated sphere of radius R moving with velocity \bar{U} , on the right we report the comparison between the velocity induced by a prescribed force \bar{F} and the free space velocity of a single sphere of radius R subject to the same external force. We let ρ vary from 2.2 to 8. The decay of the drag as ρ increases (or equivalently the increase in velocity at a given force) illustrates the phenomenon of hydrodynamic screening. The very good agreement between benchmarks and our results proves that the present method is able to reproduce properly the interaction of two simple rigidly moving bodies.

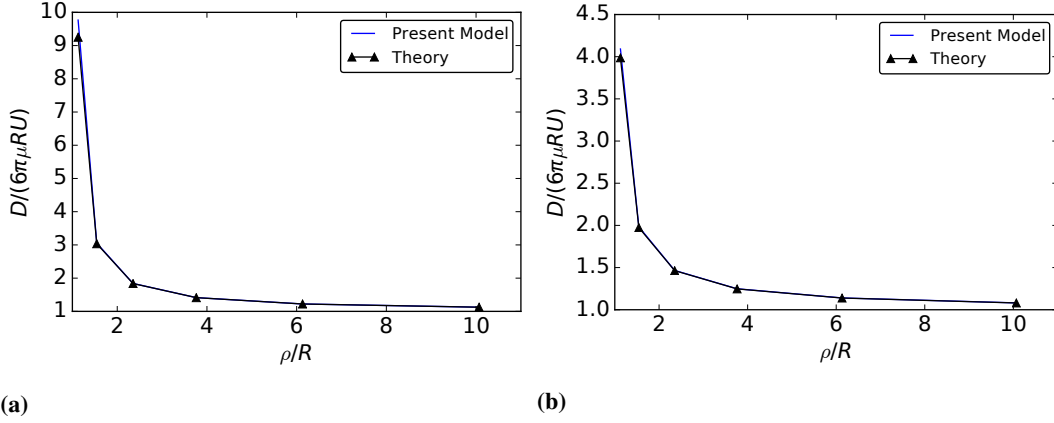


Figure 6.13: Drag analysis for a sphere towed near an interface. Figure 6.13a represents the analysis considering a single no slip wall, while Figure 6.13b shows the analysis for a single perfect slip interface. In blue with circles we plot the results of the present analysis while in black with triangle we represent the theoretical results, see [53].

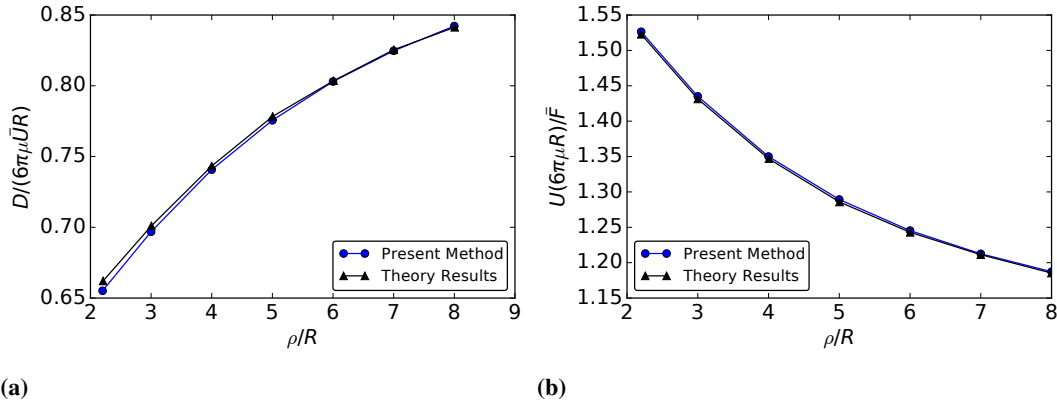


Figure 6.14: Motility Analysis for the 2 sphere system. In Figure 6.14a we compare the drag on each sphere induced by an overall unitary velocity along the centerline of the two sphere system with the drag of a single towed sphere. In Figure 6.14b we show the comparison between the velocity induced by an overall unitary force along the centerline acting on the two sphere system with the velocity of a single towed sphere. In blue with circles we plot the results of the present analysis while in black with triangle we represent the theoretical result found in [53].

6.4.4 Composite model swimmer

We model and study the complete composite bacterium made by a spherical head and a rotating flagellum as proposed in [90,91,97,109]. This kind of bacterium is considered a benchmark as reported by [109]. We consider the flagellum as a circular helix with circular cross-section of radius r and axis given by

$$\mathbf{r} = (x, y, z) = (x, bE(x)\cos(kx - \omega t), bE(x)\cos(kx - \omega t)), \quad (6.46)$$

$$E(x) = 1 - e^{-(k_E x)^2}, \quad (6.47)$$

where k_E determines how quickly the helix grows to its prescribed amplitude b . At $x = x_e = 2/k_E$ we have that $E = 0.98$, thus when $x > x_e$ the wave parameters are essentially constant, we call this region the linear region. We assume the pitch of the

helix $\lambda = 2\pi/k$, the flagellum has a total number of turns of N_λ . We consider the body of the bacterium as a sphere of radius R . We sketch a possible geometry considering $N_\lambda = 2$, $R = 1$, $b = \lambda/2/\pi$, $\lambda = 2.5$ in Figure 6.15. The following data are fixed:

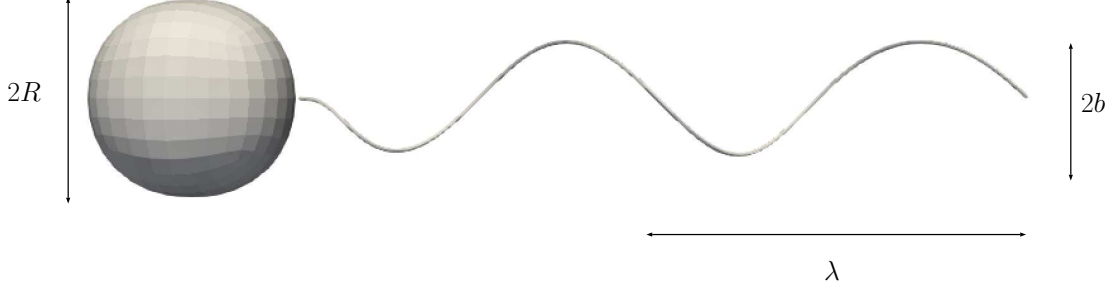


Figure 6.15: Geometry for the Bacterium as reported in [90]. We consider a spherical head of radius R and a flagellum modelled as a cylinder of radius r mapped on a helix of wavelength λ and amplitude b satisfying (6.46). As an example we consider $N_\lambda = 2$, $R = 1$, $b = \lambda/2/\pi$, $\lambda = 2.5$.

$R, L/R, r/R, bk = 1, k/k_E = 1, N_\lambda$. We need to compute the following parameters: b, k . We know that

$$L = f(\lambda) = \int_0^{\lambda N_\lambda} \left(1 + \left(\frac{\partial r_y}{\partial x} \right)^2 + \left(\frac{\partial r_z}{\partial x} \right)^2 \right) dx, \quad (6.48)$$

therefore we retrieve λ as the root of the following non linear equation

$$\bar{L} - f(\lambda) = 0. \quad (6.49)$$

Then we compute

$$k = 2\pi/\lambda, \quad (6.50)$$

$$b = \lambda/(2\pi). \quad (6.51)$$

We import the spiral, as a curve, into the computational setting and perform the simulations. We compare the results of the developed method to literature benchmarks for the considered bacterium geometry. In Section 6.4.4 we study a free space swimmer using two different analyses, the first one concerns the mean velocity along a stroke and the second one involves the instantaneous velocity at different relative rotations of the flagellum with respect to the head. We repeat such analysis for a bacterium near a no slip interface and we compare our results with [97]. Finally we consider a bacterium swimming near a perfect slip interface as described in [91].

Bacterium in free space We compare the overall free space motion along the longitudinal axis for the corkscrew bacterium. We take as reference the results reported in [90]. We refer to the geometry of Figure 6.15 considering $L/R = 20, L/R = 10, L/R = 5$, and we use $\lambda/(2\pi/\omega)$ to retrieve an adimensional velocity. We vary the number of turns per flagellum N_λ . We project the rigid velocity on the direction of the angular velocity, namely

$$U = \mathbf{V} \cdot \frac{\omega - \boldsymbol{\Omega}}{|\omega - \boldsymbol{\Omega}|}, \quad (6.52)$$

where \mathbf{V} represents the swimmer rigid linear velocity and $\omega - \Omega$ represent the actual absolute angular velocity of the flagellum. We plot the results in Figure 6.16 in red for $L/R = 20$, in blue for $L/R = 10$ and in green for $L/R = 5$. From Figure 6.16

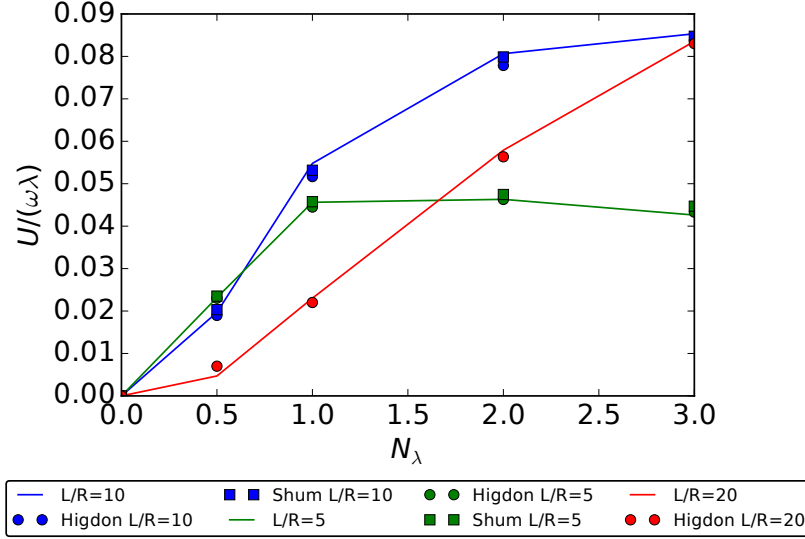


Figure 6.16: Free Space Movement Analysis. The continuous lines present the results obtained using the present method, in green we plot the results for a tail length $L = 5\lambda$, in blue we draw the results for a flagellum length $L = 10\lambda$, in red we depict the results for a tail length $L = 20\lambda$. The dots represents the analytical results by Higdon while the squares the numerical result seen in [109].

we see that we obtain a good agreement for all three flagellum lengths considered. To assess the performance of our methodology we compare the instantaneous velocities with [97]. We group the geometrical configuration of the present simulation as $R = 1$, $N_\lambda = 1.5$, $L/A = 10$, $k = k_e = 1/b$. Figure 6.17a reports the comparison for the linear velocities while in Figure 6.17b we represent the analysis for the angular velocities. We see a very good agreement between the proposed method and [97].

Bacterium near a no slip interface We study the interaction between a bacterium and a physical interface. The presence of the interface eliminates many of the residual symmetries of the previous free space micro-swimming analysis. In [97] the authors report an analysis of the instantaneous velocities of the bacterium during the stroke near a no slip wall. We analyze the results regarding a bacterium placed at a minimum distance s_d with respect to the wall

$$s_d = 0.1R, \quad (6.53)$$

where R is the radius of the bacterium body. In this particular case we are modeling a bacterium swimming parallel to the interface and the center of its body is placed at distance

$$s = 1.1R. \quad (6.54)$$

We report the comparison between the present method and [97] in Figure 6.18. We see a good agreement between the present method and [97]. We have obtained the results approximating the infinite no slip plane using a computational box of size $L \gg R$ with the upper surface representing the no slip interface. We need a refined grid to recover

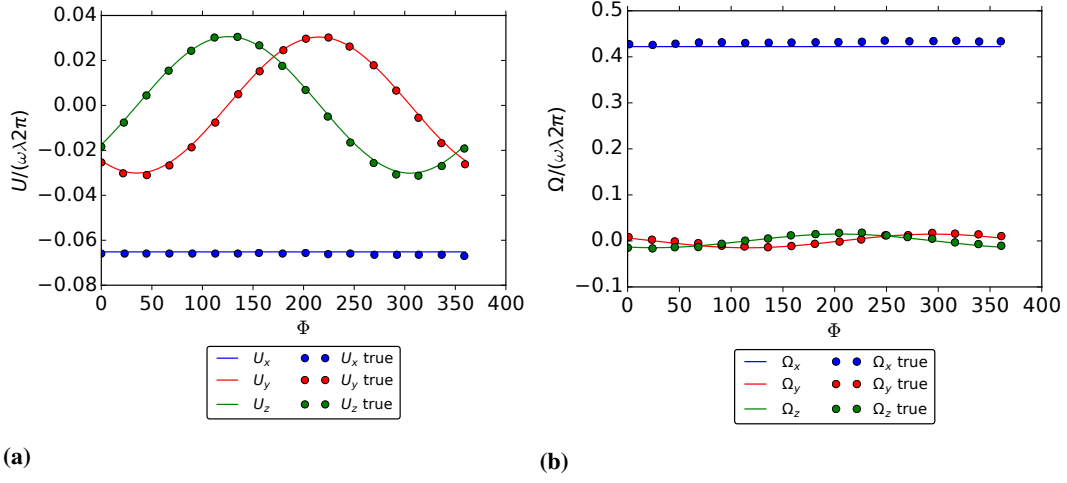


Figure 6.17: Free Space Instantaneous velocities Analysis. Figure 6.17a reports the analysis for the translational velocities, Figure 6.17b shows the angular velocities. The continuous lines represent the sinusoidal approximation starting by 8 different computations, the dots depicts the reference results by [97]. Blu, red and green plots represent the x , y , z components respectively.

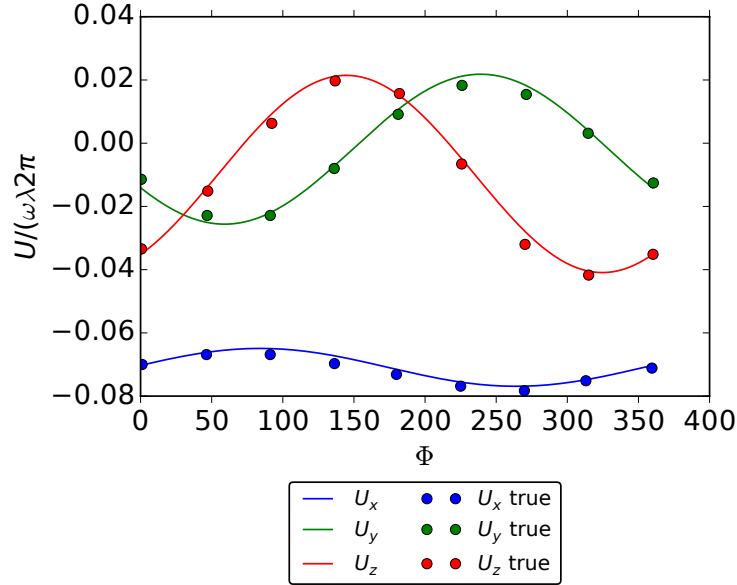


Figure 6.18: Instantaneous Velocity analysis near a no slip interface at minimum distance $h = 0.1$. The continuous lines represent the sinusoidal approximation starting by 8 different computations, the dots depicts the reference results by [97]. Blue, red and green plots represent the x , y , z components respectively.

all the peculiarities of the velocity field if the wall is very near to the swimmer. The presented results have been obtained using 12000 degrees of freedom.

Bacterium near a perfect slip interface We follow [91] to analyze the interaction between the composite swimmer and a perfect slip interface. We use the kernel for a single perfect slip interface derived in Section 5.3.1. We consider Figure 7 from [91] and we use it as a benchmark for our method. We perform three different analyses in Figures

6.19, 6.20, 6.21. We compare both the curvature radius R obtained as i

$$R = \frac{|\hat{V}|}{\hat{\Omega}_y - \hat{\Omega}_x \tan(\theta)}, \quad (6.55)$$

and the drift angle α

$$\alpha = \arcsin \left(\frac{\hat{V}_z}{\hat{V}_x} \right), \quad (6.56)$$

where $\hat{\cdot}$ represents the mean over a stroke, and θ is the pitch angle around the z axis. We consider 12 different configuration per stroke. To recover the trajectory in the phase plane θ, h we have integrated the mean velocities $\hat{\Omega}_z, \hat{V}_y$ with a low-storage Runge Kutta method.

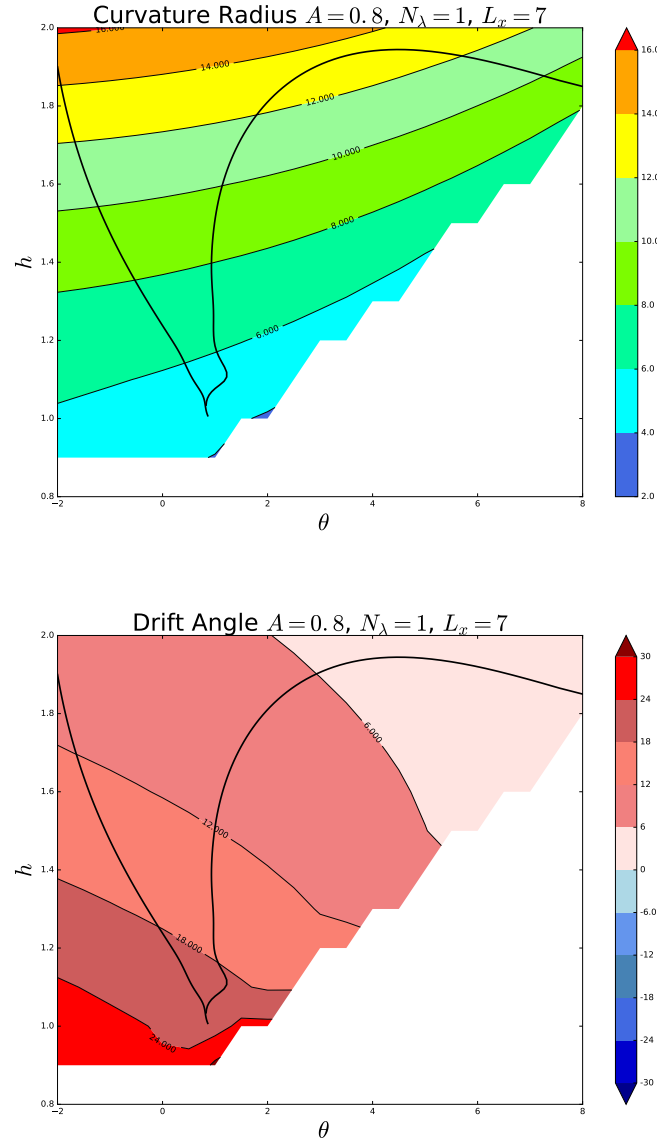


Figure 6.19: Swimming analysis of a corkscrew bacterium near a perfect slip interface. The geometric parameters of the tail are: helix amplitude $A = 0.8$, number of turns $N_\lambda = 1$, length over the x axis $L_x = 7$. On the top the curvature radius analysis, on the bottom the drift angle analysis.

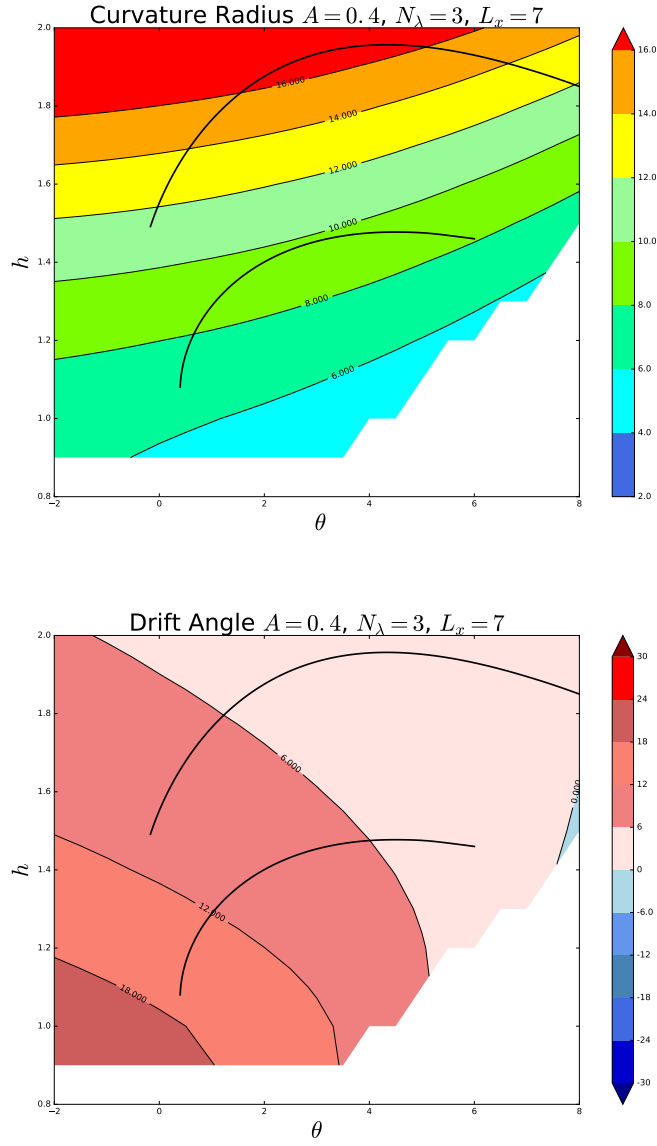


Figure 6.20: Swimming analysis of a corkscrew bacterium near a perfect slip interface. The geometric parameters of the tail are: helix amplitude $A = 0.4$, number of turns $N_\lambda = 3$, length over the x axis $L_x = 7$. On the top the curvature radius analysis, on the bottom the drift angle analysis.

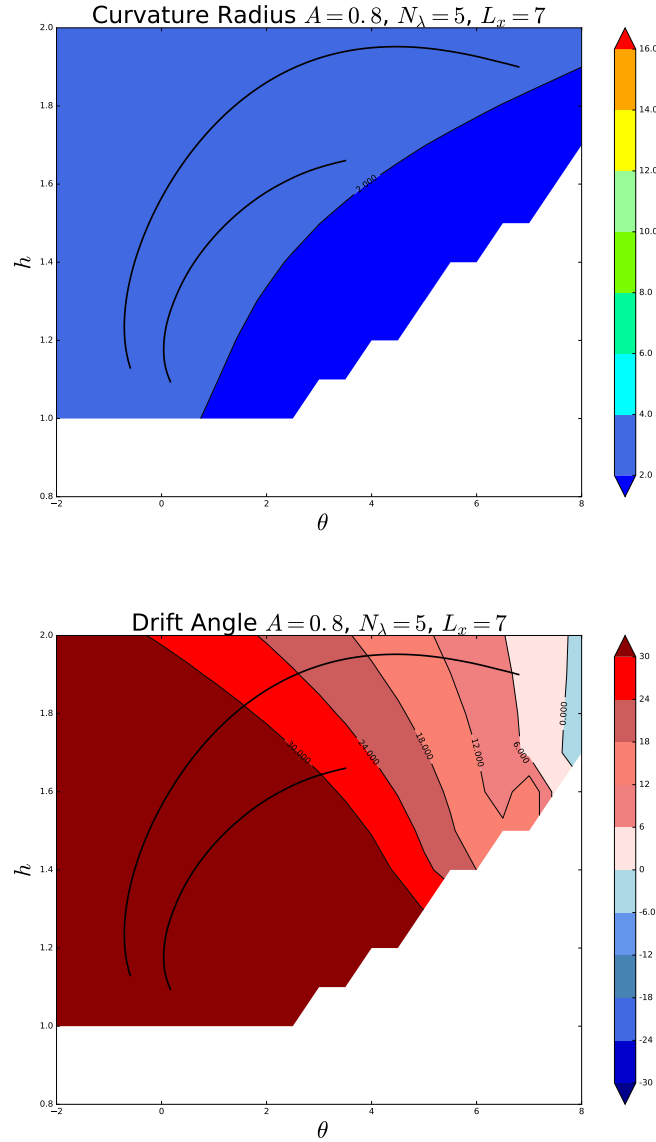


Figure 6.21: Swimming analysis of a corkscrew bacterium near a perfect slip interface. The geometric parameters of the tail are: helix amplitude $A = 0.8$, number of turns $N_\lambda = 5$, length over the x axis $L_x = 7$. On the top the curvature radius analysis, on the bottom the drift angle analysis.

We see a good agreement with all three different configurations. In our opinion the remaining differences are due to geometrical uncertainties. We perform another simulation considering a slightly different spiral for Figure 6.20. In particular we consider a spiral built following [97] so that it has $L_x = 7$ and $A = 0.37$. The results are reported in Figure 6.22. We see that the results are deeply influenced by the helical amplitude, as is expected [91].

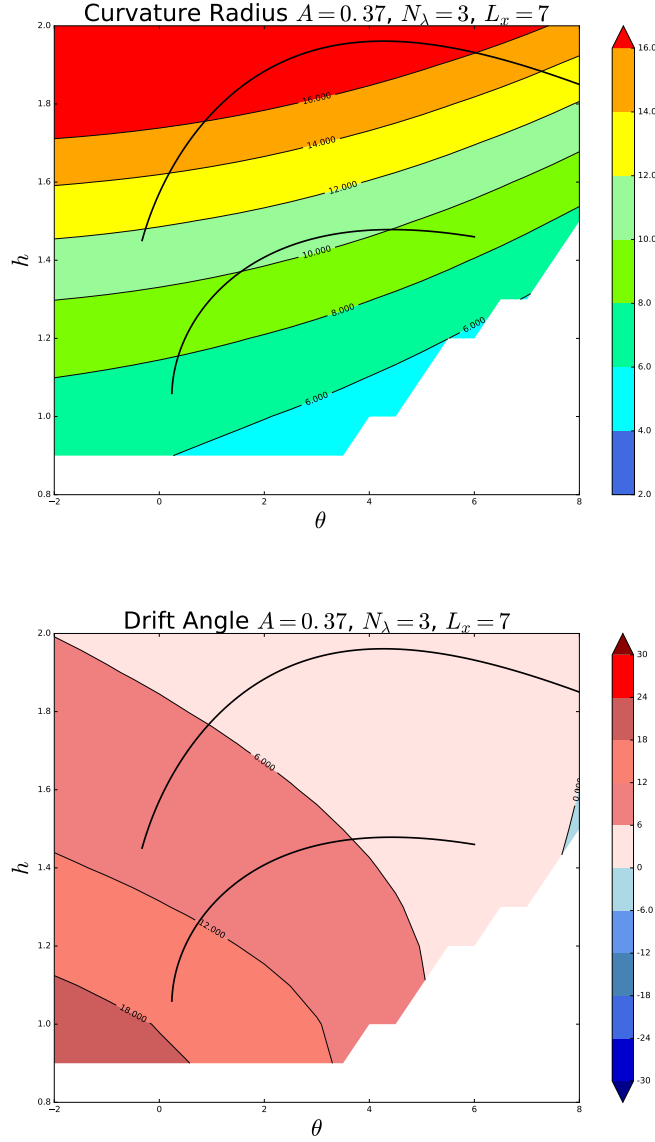


Figure 6.22: Swimming analysis of a corkscrew bacterium near a perfect slip interface. The geometric parameters of the tail are: helix amplitude $A = 0.37$, number of turns $N_\lambda = 3$, length over the x axis $L_x = 7$. On the top the curvature radius analysis, on the bottom the drift angle analysis.

6.5 Experimental application

We analyze the results of a three dimensional simulation using the images from [50] as a source for the experimental data. Figure 6.6 reports the flagella configuration used in the present Section. We recall that the shape of the flagellum at each time step is a NURBS curve obtained from a constrained least square approximation of the tracking of the original experimental setting. In [50] the authors suggest that the motion of the swimmer in a thin soap film present some clear bidimensional characteristics. For this reason we perform a simulation using a two-dimensional BEM, we plot the

results of such simulation in Figure 6.23. We clearly see that the two-dimensional

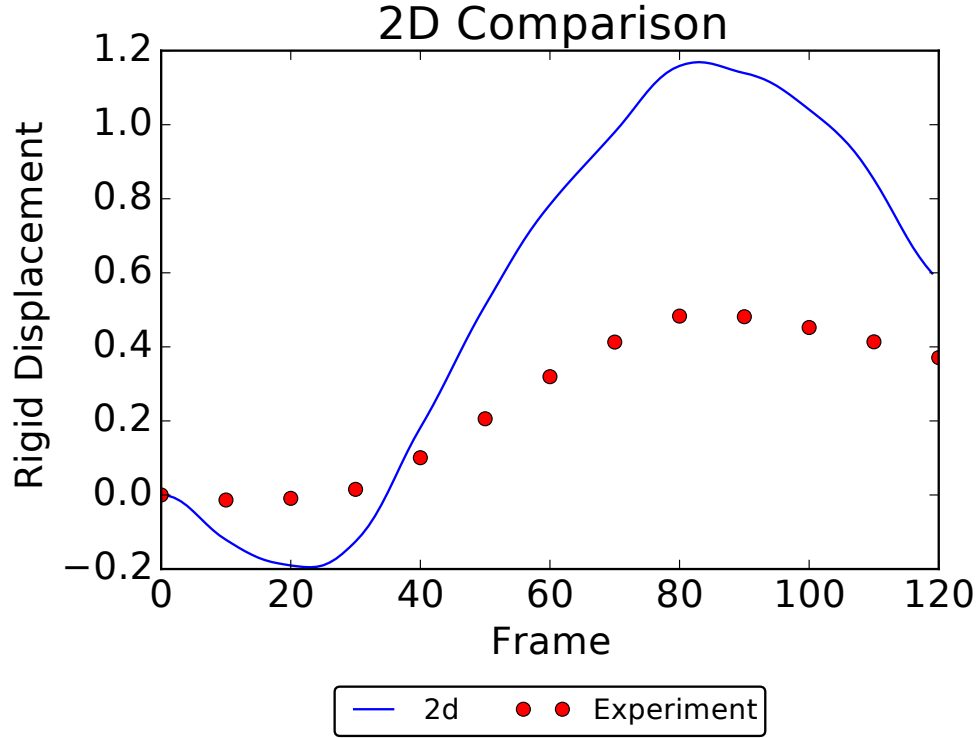


Figure 6.23: The comparison in the rigid displacements. The red dots represent the actual tracked motion. In blue we see the 2D result.

simulation overestimate both the backward and forward movements of the swimmer. A pure bidimensional simulation depicts a swimmer having an infinitively long third dimension, in such a case the flow would be bidimensional for symmetry reasons. We compare different three dimensional simulations to retrieve the three-dimensionality of the motions together with the bi-dimensionality induced by the thin film and highlighted in [50]. We report the results in Figure 6.24. We use a full three dimensional kernel, the results are plotted in blue and clearly demonstrate that the stroke maintain a clear three-dimensionality. We apply two different boundary conditions to the interfaces. We use classical no slip conditions. The results in green clearly show that such walls can't be represented using no slip boundary conditions. Then we use the perfect slip condition. We plot the results of the simulation in black and we see that the perfect slip condition approximates quite well the thin soap film. Finally we have modified the Green kernels to obtain an automatic fulfillment of the two perfect slip conditions, as described in 5.3.2. We plot the results in yellow and we see that they are comparable with the ones obtained using two perfect slip interfaces. We see that both the simulation using the two perfect slip walls and the one using the repeated kernel are able to obtain very similar, and quite good, results. This is due to the fact that the experiment is performed through the use of a thin soap film. The hydrodynamic interactions cause some bi-dimensional effects that the full three dimensional simulation is not able to reproduce.

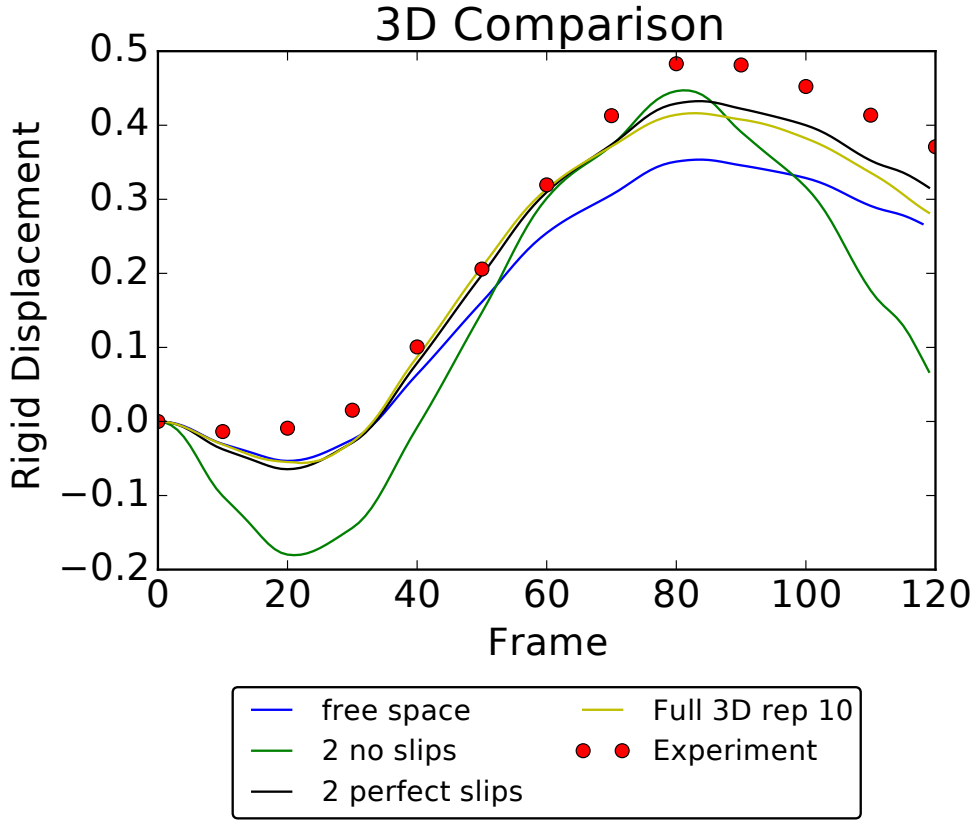


Figure 6.24: *The comparison in the rigid displacements. The red dots depicts the real tracked motion. In blue we plot the full three dimensional simulation. In green we report the result using two no slip interfaces, black using two perfect slip walls, while in yellow we have used the modified Green kernels.*

6.6 Head-Tail interactions in a model “robotic” bacterium

We now move to the study of hydrodynamic interactions in a model swimmer made by assembling distinct body parts. As a test case, we consider a “robotic” corkscrew bacterium composed of a rigid head and a rotating, rigid, helical flagellum. The head is a sphere of radius R , and the flagellum is a circular helix such as the one presented in Section 6.4.1, so that we can take advantage of the data presented in [100]. Figure 6.25 reports a sketch of the composite swimmer. By varying the length of the flagellum at fixed head size, we study the significance of hydrodynamic interactions between head and flagellum. We follow [95,96], where the author studied the motion of the composite system (head and flagellum) trying to infer its performance from the knowledge of the hydrodynamics of the separate components (body and propeller). We call such methodology the “additive approximation” or “additive approach”.

6.6.1 Optimal linear velocity in additive approach

We search for a number of turns N_λ maximizing the linear velocity of the bacterium, we follow [96] and we apply the additive approach. We write the momentum balance

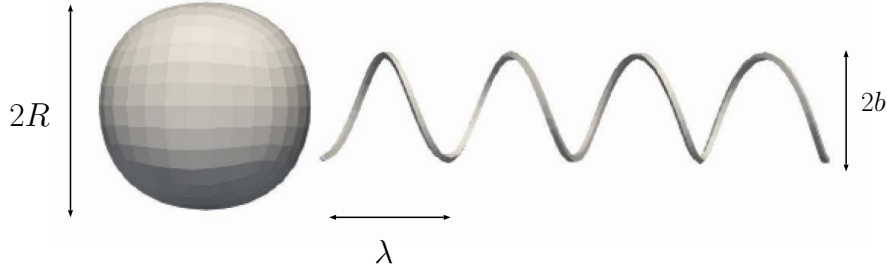


Figure 6.25: *Reference Configuration for the Bacterium.* We consider a helicoidal flagellum of amplitude b and a spherical head of radius $R = 2b$. We fix the wavelength of the flagellum $\lambda = 2.42b$ and its radius $r = b/16$. We let the number of turns N_λ vary from 1 to 20. In the figure we have as example considered $N_\lambda = 4$.

laws in AA using the superposition principle as

$$F_{tot} = F_{body} + F_{flagellum} = 0. \quad (6.57)$$

Both head and tail experience a rigid motion characterized by a linear and an angular velocity U, Ω respectively, moreover the flagellum has a relative angular velocity ω with respect to the head. We write, exploiting the concept of resistance matrix, the linear and angular momentum balances as

$$\begin{bmatrix} A_0 & \\ & C_0 \end{bmatrix} \begin{bmatrix} U \\ \Omega \end{bmatrix} + \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} U \\ \Omega - \omega \end{bmatrix} = 0. \quad (6.58)$$

where A_0, C_0 represent the drag and torque coefficients for the head, A, B, C are the drag, coupling and torque coefficients for the propeller, U, Ω are the unknown linear and angular velocities of the head while $\omega = \Omega_{head} - \Omega_{flagellum}$ represents the prescribed relative angular velocity of the helical flagellum with respect to the head. Solving for U and Ω we obtain

$$U = \frac{C_0 B}{(C_0 + C)(A_0 + A) - B^2} \omega, \quad (6.59a)$$

$$\Omega = \left(1 - \frac{(A_0 + A)C_0}{(C_0 + C)(A_0 + A) - B^2} \right) \omega. \quad (6.59b)$$

It is common to simplify (6.59a) considering

$$C \ll C_0, \quad (6.60a)$$

$$B^2 \ll C_0, \quad (6.60b)$$

in such cases we derive a simplified expression for the velocity as

$$U = \frac{B}{A_0 + A} \omega, \quad (6.61)$$

see [96] for further details. However (6.60a) does not hold true if the flagellum is long enough. In such cases we easily have $C \sim 0.3C_0$ and this clearly contradicts (6.60a). Given the linearity of the Stokes system, and the results of Figure 6.12, we assume the

coefficient A, B, C to have a linear dependence with respect to the flagellum length. Namely we assume

$$A = m_A x + q_A, \quad (6.62a)$$

$$B = m_B x + q_B, \quad (6.62b)$$

$$C = m_C x + q_C, \quad (6.62c)$$

we compute the coefficients in (6.62) by means of a linear regression of the data reported in Figures 6.12a, 6.12b, and 6.12c. Figures 6.26a, 6.26b report such approximations. Substituting (6.62) into (6.59a) we derive an analytical estimate of the translational velocity

$$U = f(x) = \frac{\alpha x + \beta}{\gamma x^2 + \delta x + \epsilon} \omega \quad (6.63)$$

where

$$\alpha = C_0 m_B, \quad (6.64a)$$

$$\beta = C_0 q_B, \quad (6.64b)$$

$$\gamma = m_A m_C - m_B^2, \quad (6.64c)$$

$$\delta = (q_A + A_0) m_C + (q_C + C_0) m_A - 2 m_B q_B, \quad (6.64d)$$

$$\epsilon = (q_A + A_0)(q_C + C_0) - q_B^2. \quad (6.64e)$$

We determine that $f(0) = \frac{\beta}{\epsilon} > 0$ and $\lim_{x \rightarrow \infty} f(x) = 0$, we consider the first derivative of (6.63) to determine the presence of a maximum for the function $f(x)$, namely we search the roots of

$$f'(x) = \frac{df}{dx} = \frac{-\alpha \gamma x^2 - 2\beta \gamma x + \epsilon \alpha - \beta \delta}{(\gamma x^2 + \delta x + \epsilon)^2} \omega. \quad (6.65)$$

We see that $f'(x) = 0$ admits two different solutions,

$$x_{1,2} = \frac{\beta \gamma \pm \sqrt{\beta^2 \gamma^2 \alpha \gamma (\epsilon \alpha - \beta \delta)}}{\alpha \gamma}, \quad (6.66)$$

and considering (6.64) we determine $x_2 < 0$, therefore

$$x_{max} = \frac{\beta \gamma + \sqrt{\beta^2 \gamma^2 \alpha \gamma (\epsilon \alpha - \beta \delta)}}{\alpha \gamma}. \quad (6.67)$$

In Figure 6.26d we report the comparison between the effective results of the method considering the two resistance matrices for the spherical head and the flagellum and the two expressions (6.59a), (6.61) considering the linear regressions (6.62). From Figure 6.26d we see that (6.63) (green curve) recovers very well the behavior of the velocity (blue with circles), and that the maximum point (6.67) (black diamond) well represents the actual maximum of the curve coming from the solution of the linear system (6.58).

6.6. Head-Tail interactions in a model “robotic” bacterium

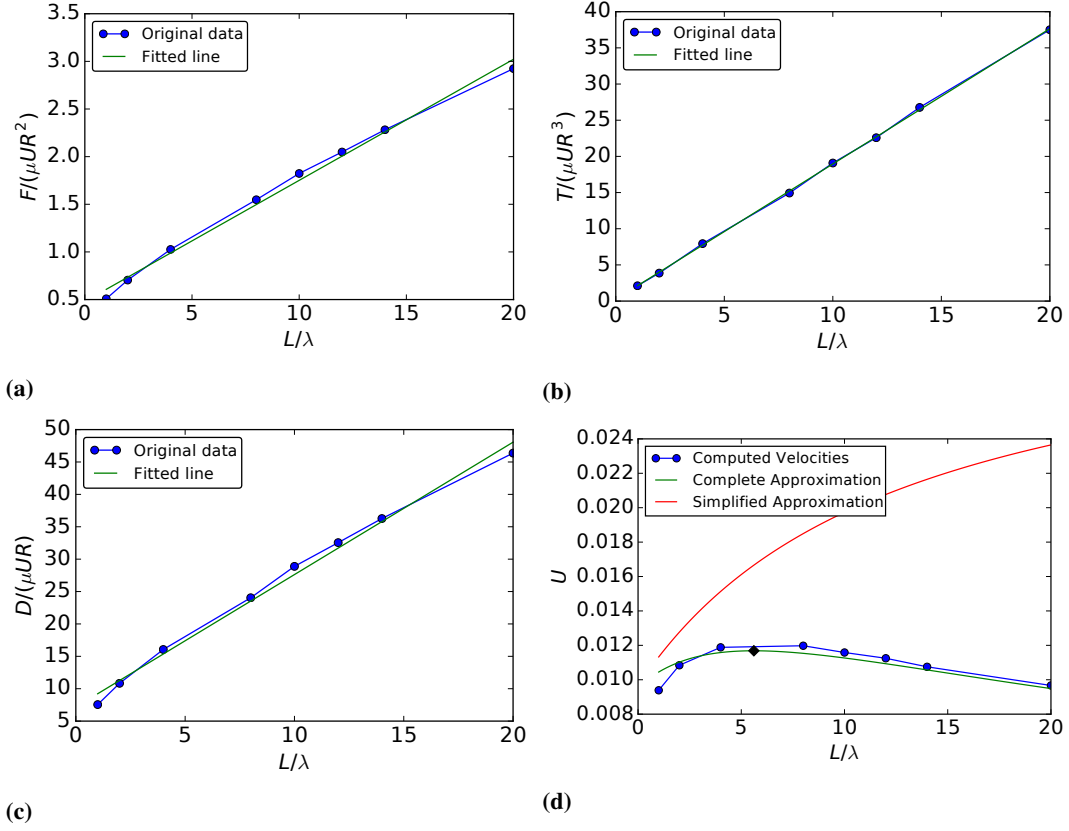


Figure 6.26: Analysis for the linear regression of the resistive coefficients. Figure 6.26a represents the linear regression for the force coefficient, in blue we plot the original data computed using BEM and in green we draw the corresponding linear regression. Figure 6.26b represents the linear regression for the torque coefficient, in blue we plot the original data computed using BEM and in green we draw the corresponding linear regression. Figure 6.26c represents the linear regression for the drag coefficient, in blue we plot the original data computed using BEM and in green we draw the corresponding linear regression. In Figure 6.26d we compare the analytical solution obtained using the linear regression for the coefficients (6.62) and the computed velocity. In blue we plot the computed results, in green we draw the exact approximation (6.59a) and in red the simplified approximation (6.61).

6.6.2 Additive vs Global approach for linear and angular velocity

We compare the results obtained using the additive approach (AA) with the accurate resolution, via BEM, of the hydrodynamics of the entire robotic bacterium. We call the latter “global approach” (GA). It is expected from [96] that, as the length of the flagellum increases, the error induced by neglecting head-tail interactions should decrease. Thus we let the number of turns N_λ vary from 1 to 20, keeping λ and ω fixed, and we compare both the angular velocity Ω and the swimming speed U obtained with the two approaches. In Figure 6.27 we report the comparison between AA and GA for the angular velocity Ω : blue circles in Figure 6.27a show the solution obtained with the global approach while green squares show the results obtained with the additive approach. Figure 6.27b represents the relative error introduced by the additive approach. In Figure 6.28 we compare the results for the swimming speed U . Blue circles in Figure 6.28a show the GA results and green squares are relative to the solution obtained with AA.

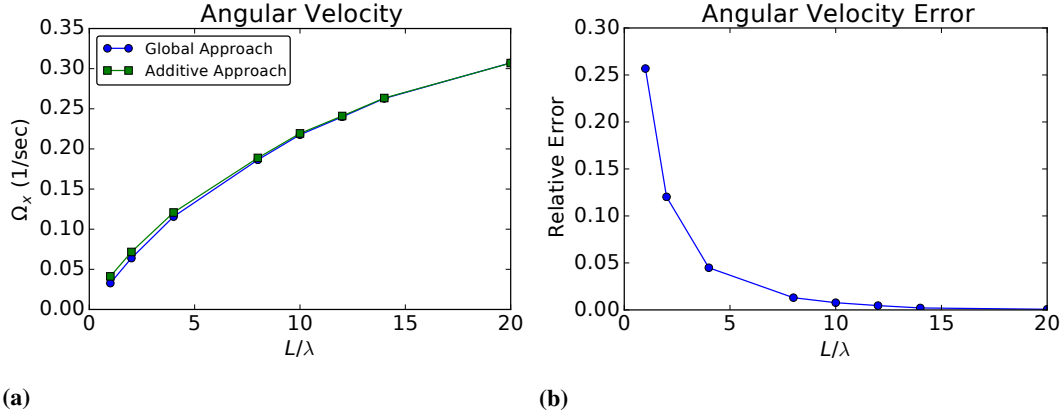


Figure 6.27: Angular velocity comparison. In Figure 6.27a we report the angular velocity obtained using the global approach (blue circles), and solution obtained with the additive approximation which neglects the interactions between body and flagellum (green squares). In Figure 6.27b we plot the relative error introduced by the additive approach.

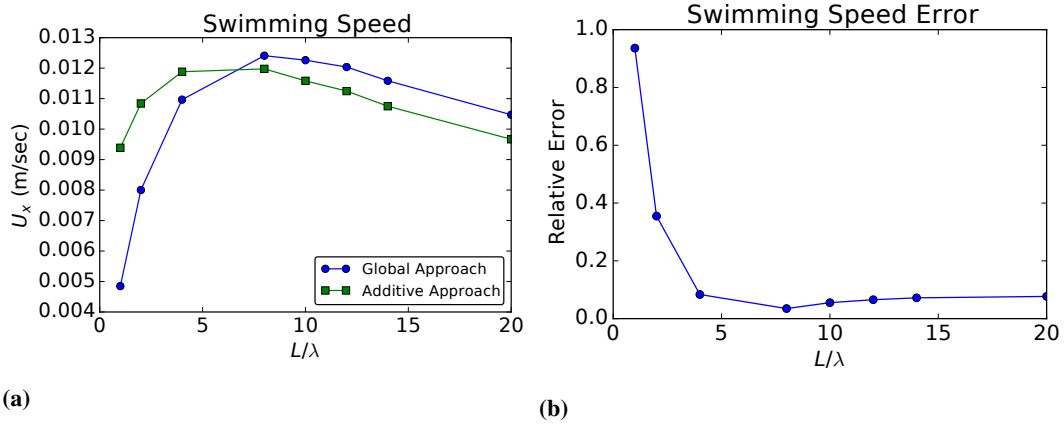


Figure 6.28: Swimming speed comparison. In Figure 6.28a we report the swimming speed obtained using the global approach (blue circles), and solution obtained with the additive approximation which neglects the interactions between body and flagellum (green squares). In Figure 6.28b we plot relative error introduced by the additive approach.

In Figure 6.28b we plot instead the relative error. We see that, for what concerns the angular velocity, the additive approximation does not introduce significant errors. Moreover, such errors decrease quickly as the relative length of the flagellum increases with respect to the head size. However, for what concerns the swimming speed, the error is never negligible for any of the configurations considered: it is very significant for short flagella (small N_λ), and it stabilizes at a relative small value ($< 10\%$) for $N_\lambda > 5$. While both the approaches lead to a maximum in the velocity, these maxima are observed for different values of N_λ .

6.6.3 A simple formula provides a correction for the additive approach

We want to understand the differences in swimming speed U computed using either the additive approximation or the global approach for the entire bacterium. In Section 6.6.1 we obtained, following [96], the momentum balance laws for the swimmer as (6.58).

We recall that solving for U and Ω we obtain

$$U = \frac{C_0 B}{(C_0 + C)(A_0 + A) - B^2} \omega, \quad (6.68a)$$

$$\Omega = \left(1 - \frac{(A_0 + A)C_0}{(C_0 + C)(A_0 + A) - B^2} \right) \omega. \quad (6.68b)$$

If we consider the hydrodynamics of the entire system, without invoking the additive approximation, we write the velocity field of the swimmer as

$$\underline{v}(x) = U_1 \underline{e} \chi_1(x) + U_2 \underline{e} \chi_2(x) + \omega_1 \underline{e} \wedge (x - o) \chi_1(x) + \omega_2 \underline{e} \wedge (x - o) \chi_2(x), \quad (6.69)$$

where χ_1 and χ_2 are the characteristic functions of body 1 (the head) and 2 (the propeller) *i.e.*, $\chi_1(x) = 1$ if x belongs to body 1 and $\chi_1(x) = 0$ otherwise. Moreover, \underline{e} is a unit vector along the axis of the helical flagellum. Using the linearity of the Stokes system, we can write the force and torque (with respect to the pole o) acting on the whole body, respectively as

$$A_1 U_1 + \hat{B}_1 \Omega_1 + A_2 U_2 + \hat{B}_2 \Omega_2, \quad (6.70a)$$

$$\bar{B}_1 U_1 + C_1 \Omega_1 + \bar{B}_2 U_2 + C_2 \Omega_2, \quad (6.70b)$$

where A_1 is the viscous force on the whole system arising from the velocity field (6.69) with $U_1 = 1$ and $U_2 = \Omega_1 = \Omega_2 = 0$. A similar interpretation holds for the other coefficients \hat{B}_i (giving forces induced by rotation of the body i , in the presence of the other body parts kept fixed), \bar{B}_i (giving torques induced by translation of the body i , in the presence of the other body parts kept fixed), and C_i (giving torques induced by rotation of the body i , in the presence of the other body parts kept fixed). Requiring that

$$U_2 = U_1, \quad (6.71a)$$

$$\Omega_2 = \Omega_1 - \omega, \quad (6.71b)$$

and that total viscous forces and torques vanish, we obtain the system

$$\begin{bmatrix} A_1 & \hat{B}_1 \\ \bar{B}_1 & C_1 \end{bmatrix} \begin{bmatrix} U \\ \Omega \end{bmatrix} + \begin{bmatrix} A_2 & \hat{B}_2 \\ \bar{B}_2 & C_2 \end{bmatrix} \begin{bmatrix} U \\ \Omega - \omega \end{bmatrix} = 0. \quad (6.72)$$

Notice that we can rewrite (6.72) as

$$[\mathcal{R}] \begin{bmatrix} U \\ \Omega \end{bmatrix} = \begin{bmatrix} A_2 & \hat{B}_2 \\ \bar{B}_2 & C_2 \end{bmatrix} \begin{bmatrix} 0 \\ \omega \end{bmatrix}, \quad (6.73)$$

where

$$[\mathcal{R}] = [[\mathcal{R}_1 + \mathcal{R}_2]] = \left[\begin{bmatrix} A_1 & \hat{B}_1 \\ \bar{B}_1 & C_1 \end{bmatrix} + \begin{bmatrix} A_2 & \hat{B}_2 \\ \bar{B}_2 & C_2 \end{bmatrix} \right] \quad (6.74)$$

is the resistance matrix of the complete swimmer. Thus \mathcal{R} is symmetric (by reciprocity) and positive definite, even though the two summands \mathcal{R}_1 and \mathcal{R}_2 defining \mathcal{R} are not

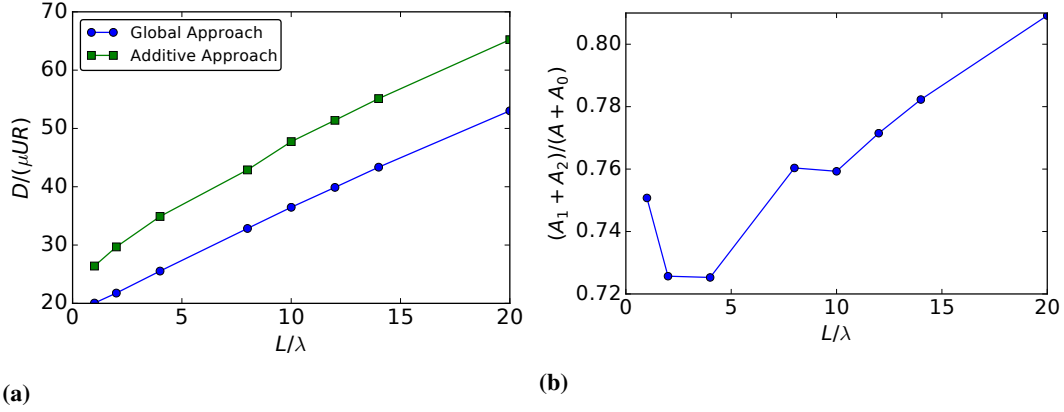


Figure 6.29: Comparison of the axial (along x) drag force induced by a longitudinal (along x) swimming speed. In Figure 6.29a we report the comparison between the coefficient obtained with the global approach (blue with circles) and sum of body and flagellum coefficients computed separately using the additive approach (green with squares). In Figure 6.29b we show ratio between the global drag and the sum of the two separate contribution of head and flagellum.

individually symmetric. Therefore $\hat{B}_1 + \hat{B}_2 = \bar{B}_1 + \bar{B}_2$ (and we will write $B_1 + B_2$ for any of these two sums) and R is invertible. Solving for U and Ω we obtain

$$U = \frac{(B_1 + B_2) \left((C_1 + C_2) \frac{\hat{B}_2}{B_1 + B_2} - C_2 \right)}{(C_1 + C_2)(A_1 + A_2) - (B_1 + B_2)^2} \omega, \quad (6.75a)$$

$$\Omega = \left(\frac{\hat{B}_2}{B_1 + B_2} - \frac{(A_1 + A_2) \left((C_1 + C_2) \frac{\hat{B}_2}{B_1 + B_2} - C_2 \right)}{(C_1 + C_2)(A_1 + A_2) - (B_1 + B_2)^2} \right) \omega. \quad (6.75b)$$

We analyze the differences between (6.68a) and (6.75a) to understand the discrepancies in Figure 6.28. In the additive approach, $A_0 + A$ and $C_0 + C$ represent the global hydrodynamic coefficients for drag and torque experienced by the whole swimmer, and the single elements A_0, A, C_0, C are coefficients for drag and torque experienced by swimmer parts considered alone in free space. In the global approach, the global hydrodynamic coefficients are given by $A_1 + A_2$ and $C_1 + C_2$, where the single components A_1, A_2, C_1, C_2 represent, as already mentioned, the drag and torque experienced by the whole swimmer induced by the movement of one of its parts, computed considering the presence of all the other parts kept fixed. In Figure 6.29a we compare the drag coefficient of the composite system (blue circles) with the sum of the drags of the single components, namely body and flagellum, computed separately (green squares). In Figure 6.29b we plot the ratio between these two quantities. Figure 6.30a compares the complete torque coefficient in the global approach (blue circles) with the sum of the coefficients of head and propeller computed using the additive approximation (green squares). In Figure 6.30b we plot the ratio between these two quantities.

As expected from hydrodynamic screening (see Section 6.4.3), which is only present in the global approach, the additive approximation always overestimates both drag and torque coefficients. The rationale behind the fact that AA gives poor results for the drag due to translations, and good ones for torque due to rotations is the following.

6.6. Head-Tail interactions in a model “robotic” bacterium

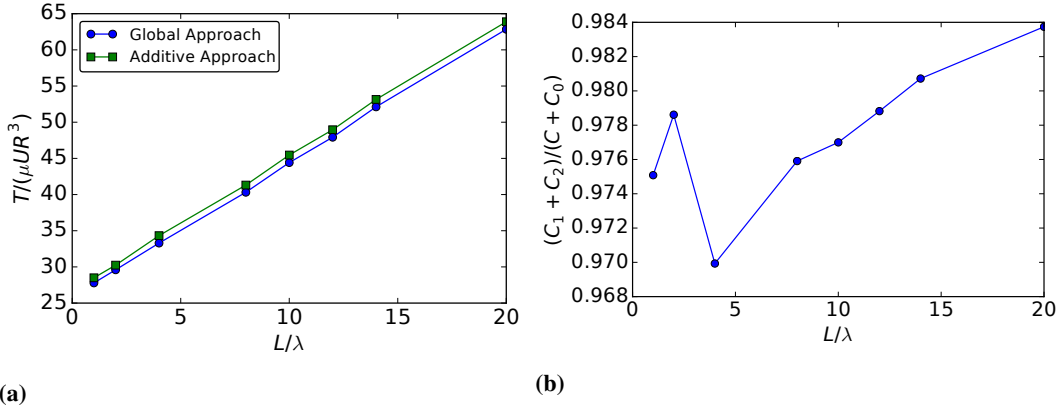


Figure 6.30: Comparison of the axial (along x) torque induced by a longitudinal (along x) angular velocity. In Figure 6.30a we show comparison between the coefficient obtained with the global approach (blue with circles) and sum of body and flagellum coefficients computed separately using the additive approach (green with squares). In Figure 6.30b we plot the ratio between the global drag and the sum of the two separate contribution of head and flagellum.

A translating sphere can be modeled as a stokeslet, with a slow (linear) decay of the induced velocity as the distance from the source increases, and the flagellum is never far enough to neglect hydrodynamic interactions. By contrast, a rotating sphere can be described as a rotlet, with a faster (quadratic) decay, and the additive approach safely estimates the overall coefficient.

The swimming speed U is also influenced by the coupling coefficients, which are different using the two different approaches. We notice that when $A_0 + A \sim A_1 + A_2$ (only approximatively satisfied when the flagellum is very long compared to the head size, $L/\lambda > 15$) and $C_0 + C \sim C_1 + C_2$ (always true), (6.75a) collapses into (6.68a) when $\hat{B}_2/B \sim 1$, $\hat{B}_2/(B_1 + B_2) \sim 1$. We study these last two conditions in Figure 6.31. Figure 6.31a shows the ratio \hat{B}_2/B and highlights the influence of the head on the flagellum coupling coefficient. In Figure 6.31b we plot $\hat{B}_2/(B_1 + B_2)$ that represents the relative importance of the flagellum coupling coefficient in the global approach. We note that the head contribution represents a minor part of the total coupling coefficient already when $N_\lambda > 2$. Thus we can safely neglect this contribution considering $\hat{B}_2 \sim (B_1 + B_2)$. However, from Figure 6.31a, we see that $\hat{B}_2 \sim B$ is approximately satisfied only when the flagellum is very long ($L/\lambda > 15$) compared to the head size, meaning that the flagellum coupling coefficient is strongly influenced by the presence of the spherical head.

Summarizing, because of the screening effect induced by the translating spherical head we have that $A_1 + A_2 \approx A_0 + A$. Moreover, the head-tail hydrodynamic interactions cause $\hat{B}_2 \approx B$ as revealed by the global approach. These two conditions provide an explanation for the error introduced by the additive approach, shown in Figure 6.28a.

A combination of the resistance coefficients introduced above provides a simple yet effective correction for the additive approach. As suggested by the previous analysis, we can consider $\hat{B}_2 \sim (B_1 + B_2)$ and $C_1 \sim C_0$, $C_2 \sim C$. Therefore, we can write (6.75a) as

$$U = \frac{(B_1 + B_2)C_0}{(C + C_0)(A_1 + A_2) - (B_1 + B_2)^2} \omega, \quad (6.76)$$

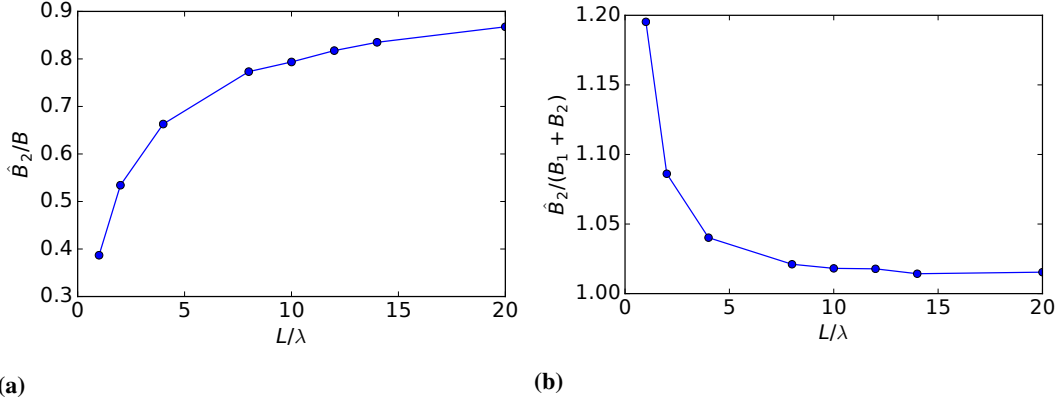


Figure 6.31: Comparison of the coupling coefficient representing the axial (along x) force induced by a longitudinal (along x) angular velocity. In Figure 6.31a we plot the ratio between the coupling coefficient due to the flagellum in the global approach and the coupling coefficient of the flagellum in free space. In Figure 6.31b we show the ratio between the coupling coefficient due to the flagellum and the complete coupling coefficient both computed using the global approach.

and we notice that $\frac{(B_1+B_2)^2}{(A_1+A_2)} \sim 0$, and $\frac{B^2}{(A+A_0)} \sim 0$ getting

$$U = \frac{(B_1 + B_2)}{(A_1 + A_2)} \frac{C_0}{(C + C_0) - \frac{(B_1+B_2)^2}{(A_1+A_2)}} \omega \sim \frac{(B_1 + B_2)}{(A_1 + A_2)} \frac{C_0}{(C + C_0)} \omega. \quad (6.77)$$

Using the same approximations we can rewrite (6.68a) as

$$U = \frac{(B)}{(A + A_0)} \frac{C_0}{(C + C_0)} \omega. \quad (6.78)$$

The ratio between equations (6.77) and (6.78) provides a correcting factor v for the swimming speed U . Namely, we write such a correction as

$$v = \frac{(A_0 + A)(B_1 + B_2)}{(A_1 + A_2)B}, \quad (6.79)$$

and we notice that v depends only on geometric parameters, *i.e.*, with our assumptions,

$$v = v \left(\frac{L}{\lambda} \right). \quad (6.80)$$

We can write

$$U_{corr} = v U = v \left(\frac{L}{\lambda} \right) \frac{C_0 B}{(C + C_0)(A + A_0) - B^2} \omega. \quad (6.81)$$

We study the correcting factor v in Figure 6.32. Figure 6.32a shows v , as a function of flagellum length, while Figure 6.32b shows a comparison among the predictions of the swimming speed computed using the global approach (blue circles), the additive approach (green squares) and the corrected additive approach (black diamonds). The correcting factor v greatly improves the accuracy of the prediction of the swimming speed that can be obtained using the additive approximation. Moreover it is very close

6.6. Head-Tail interactions in a model “robotic” bacterium

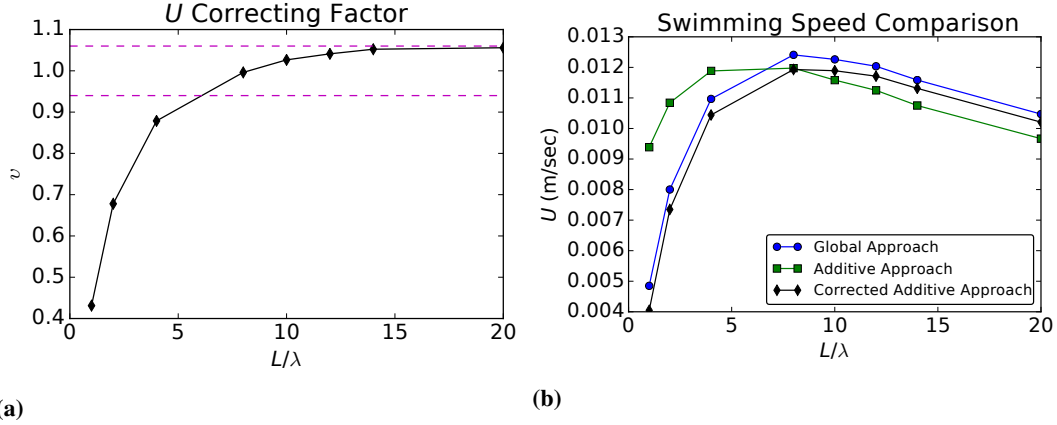


Figure 6.32: Swimming speed correction for the additive approach. In Figure 6.32a we plot the correcting factor v (black diamonds), and error thresholds corresponding to $\pm 6\%$ (dashed magenta lines). In Figure 6.32b we report the swimming speed obtained with the global approach (blue circles), solution using the additive approximation (green squares), and the results of the additive approach corrected using v (black diamonds).

(to within less than 6%) to 1 if we consider a sufficiently long tail ($L > 5\lambda$). This is consistent with [96] since a long flagellum should be well approximated by considering separately head and tail. In Figure 6.32a the dashed magenta lines show the two error thresholds corresponding to $\pm 6\%$.

The correcting factor v represents a simple way to improve *a posteriori* the results that can be obtained considering the additive approximation. In spite of its simplicity, the correction recovers the most important features of the hydrodynamic interactions between head and flagellum. In the regime of geometries ($N_\lambda < 3\lambda$ and $\lambda > 6b$) in which it is safe to use Resistive Force Theory to compute the hydrodynamic coefficients of the flagellum (see [100]), the correcting factor v makes it possible to use RFT to safely predict the performance of the swimmer. This is the main result of this paper and it is further discussed in Section 6.6.5.

6.6.4 Convergence of resistance coefficients

We investigate the convergence of the resistance coefficient for the entire bacterium $A_1 + A_2, B_1 + B_2, C_1 + C_2$, computed with the global approach, to the corresponding values $A_0 + A, B, C_0 + C$ obtained with the additive approximation when the flagellar length increases. In Figures 6.33a, 6.33b and 6.33c we plot with a blue line the drag, coupling and torque coefficient, respectively; the black dashed line shows the reference linear convergence. All the resistance coefficients show a convergence rate which is at most linear; both the drag and torque coefficients, Figures 6.33a and 6.33c, show an initial plateau followed by a linear convergence. Broadly speaking the screening effect between head and tail explains this phenomenon, but depending on which specific resistance coefficient we study, the relevant hydrodynamic interactions are different, and different trends are observed. In fact, as shown in Section 6.6.3, a dragged sphere can be modeled as a stokeslet (linear decay of the induced velocity as the distance from the source increases), while a rotating sphere is a rotlet (quadratic decay of the induced velocity as the distance from the source increases). A part of the flagellum lies in the

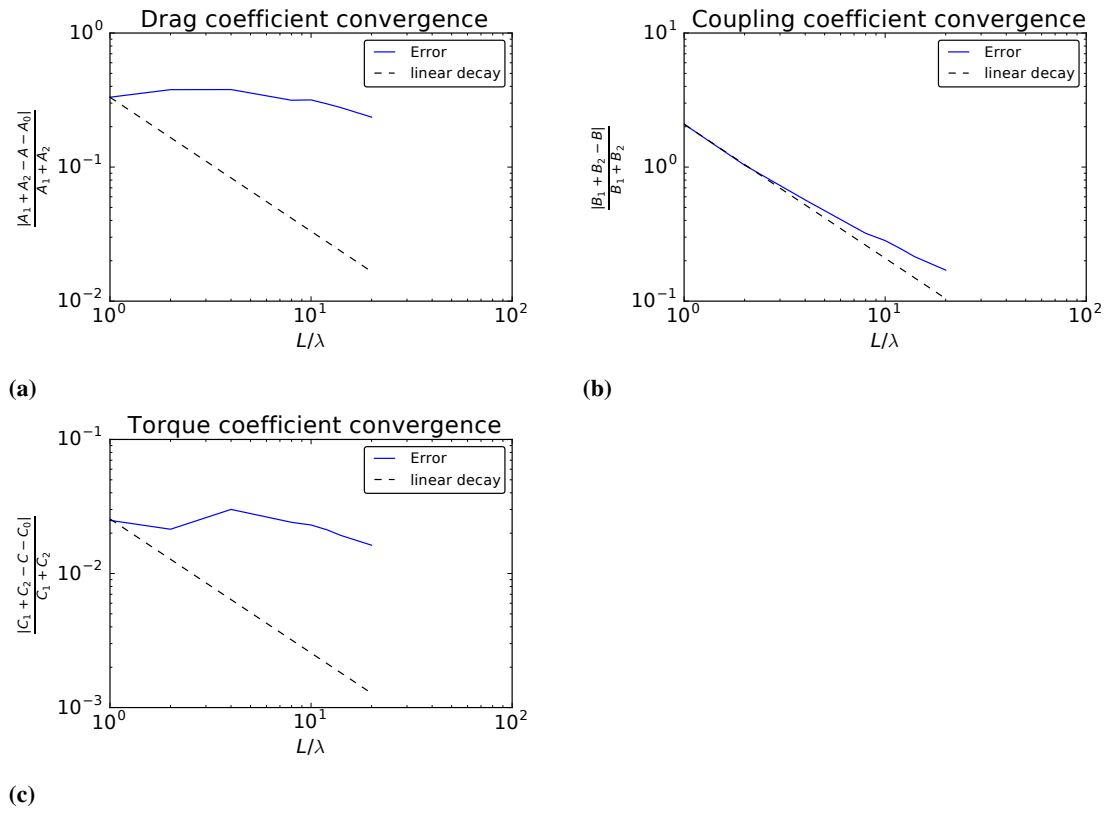


Figure 6.33: Convergence of exact coefficients obtained with the global approach to the ones computed with the additive approximation. Analysis of the convergence (blue lines) and linear reference rate (dashed black lines). Figure 6.33a: drag coefficient, Figure 6.33b: coupling effects, Figure 6.33c: torque coefficient.

6.6. Head-Tail interactions in a model “robotic” bacterium

Model	U	error
Global Approach	0.02301	
Additive Approach	0.02512	9.17
Additive Approach corrected	0.02261	1.72%
RFT Gray Hancock	0.02644	14.89%
RFT Gray Hancock corrected	0.02460	1.49%
RFT Lighthill	0.03276	42.37%
RFT Lighthill corrected	0.02284	0.74%

Table 6.1: Comparison between estimates of the swimming speed U using different approximations.

“wake” of the head and this induces a plateau in the convergence rate. In particular such plateau is smaller if we consider the torque effects thanks to the faster quadratic decay of a rotlet, and Figures 6.33c and 6.33a confirm such effect. Once the screening is fully developed, there is a finite difference between coefficients computed using the global or local approach, roughly independent of flagellar length. Hydrodynamic interactions always reduce the resistance coefficients, see Figure 6.29a and 6.30a. The resistance coefficients of the spiral alone increase, as a first order approximation (see Figure 6.12), linearly with respect to the flagellar length, therefore the convergence rate of the global approach to the additive approximation can be at most linear.

6.6.5 Correction of RFT predictions

The correcting factor v is effective even on predictions of the swimming speed U based on Resistive Force Theory (RFT). To prove this we consider, as flagellum, a circular helix with the following parameters, $a = b/16$, $\lambda = 8b$, $N_\lambda = 2$, which is a typical helix that can be well studied using RFT, see [100]. We consider again a sphere of radius $R = 2b$ as head and we estimate the error introduced both by the additive approach and by classical RFT methods. We apply such methodologies following Gray and Hancock, see [44], and Lighthill, see [69], and we use the results of the global approach as reference. Table 6.1 shows the comparison between the discussed approaches. We correct the approximated results by AA and RFT using v computed as follows: we evaluate the coefficients A_0, A, B, C, C_0 using either AA or RFT, while we compute $(A_1 + A_2), (B_1 + B_2)$ using BEM. The Table proves that v is not only able to reduce the error of the additive approach, but it can also be applied to RFT predictions reducing the relative errors by at least one order of magnitude.

Gray and Hancock derived RFT considering that the forces on an infinitesimal segment of a very slender flagellum moving at very low speed can (by analogy with those acting on an ellipsoid) be seen as directly proportional to the velocity of the segment itself and to the viscosity of the fluid. Two different proportionality constants C_N, C_T acting on normal and tangential velocity respectively, are introduced

$$C_T = \frac{2\pi\mu}{\log(2q/b) + 0.5}, \quad (6.82a)$$

$$C_N = \frac{4\pi\mu}{\log(2q/b) - 0.5}, \quad (6.82b)$$

which depend on the choice of q , Gray and Hancock assumed $q = \lambda$. In [69] Lighthill

discussed different models for the coefficients C_T, C_N , both considering different choices for the parameter ($q = 0.09\lambda$) and proposing different expressions replacing (6.82).

6.6.6 Efficiency

We want to investigate the importance of hydrodynamic interactions in the computation of the efficiency of micro-swimmers. Several notions of efficiency exist, and we consider here two kinds: energetic efficiency and swimming efficiency. The first one is the ratio between useful work rate performed by the system and total power input in the system. The second one is the net displacement in one cycle, *i.e.* a normalization of the swimming velocity.

Energetic Efficiency The input power is the one expended by the motor, and is the product of the torque acting on the flagellum multiplied by the relative angular velocity ω . There are different choices for the useful work rate. Following [90, 96], one option is the power expended to move the head at velocity U , so that

$$\eta_{en,1G} = \frac{D_{head}U}{T_{motor}\omega} = \frac{A_0U^2}{T_{motor}\omega}. \quad (6.83)$$

Here $D_{head} = A_0U$ is the drag experienced by the head, which contains the payload of our robotic swimmer, and thus is the only term allowed to contribute to the useful work rate. Using the additive approximation, we can write (6.83) as

$$\eta_{en,1A} = \frac{A_0C_0B^2}{((A + A_0)C - B^2)((A + A_0)(C + C_0) - B^2)}. \quad (6.84)$$

Another possibility takes into account the drag of the entire bacterium, see [62], namely

$$\eta_{en,2G} = \frac{D_{total}U}{T_{motor}\omega} = \frac{(A_1 + A_2)U^2}{T_{motor}\omega}, \quad (6.85)$$

where $(A_1 + A_2)U$ is the drag experienced by the whole swimmer, consisting of both the head (the payload) and the flagellum (its propulsive apparatus). Using the additive approximation this can be expressed as

$$\eta_{en,2A} = \frac{(A + A_0)C_0B^2}{((A + A_0)C - B^2)((A + A_0)(C + C_0) - B^2)}. \quad (6.86)$$

We favour $\eta_{en,1}$ over $\eta_{en,2}$ as the “correct” notion of energetic efficiency. But separating the (useful) work done to push the payload from the (passive) one needed to push the propeller without neglecting hydrodynamic interactions requires some care.

We use the factor v , computed using (6.79), to correct the energetic efficiencies. Since both (6.84) and (6.86) depend quadratically on the swimming velocity U , we introduce a corrected additive approach as

$$\eta_{en,1v} = v^2\eta_{en,1A} = v^2 \frac{A_0C_0B^2}{((A + A_0)C - B^2)((A + A_0)(C + C_0) - B^2)}, \quad (6.87)$$

and

$$\eta_{en,2v} = v^2\eta_{en,2A} = v^2 \frac{(A + A_0)C_0B^2}{((A + A_0)C - B^2)((A + A_0)(C + C_0) - B^2)}. \quad (6.88)$$

6.6. Head-Tail interactions in a model “robotic” bacterium

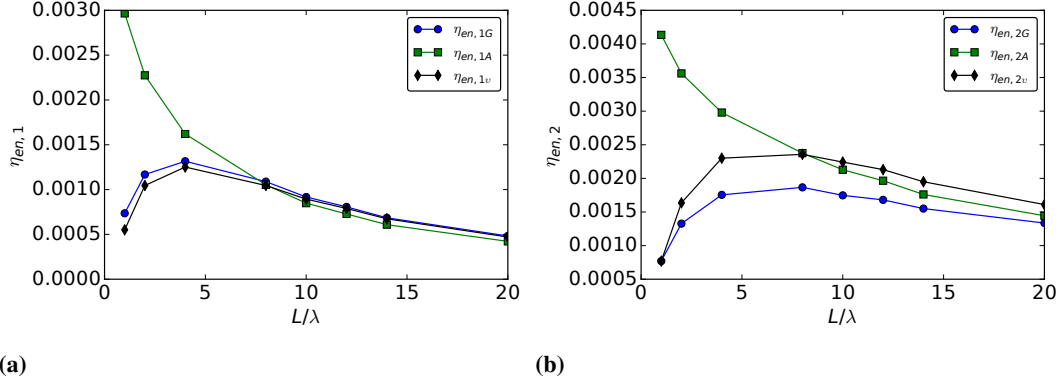


Figure 6.34: Analysis of the energetic efficiencies. Comparison between global approach (blue circles), local approximation (red triangles) and corrected local approach (green squares). In Figure 6.34a we report the analysis of the energetic efficiency $\eta_{en,1}$. In Figure 6.34b we show the comparison of the energetic efficiency $\eta_{en,2}$.

Figure 6.34 compares the energetic efficiencies computed using the additive approximation (red triangles), the global approach via BEM (blue circles), and the corrected additive approach (green squares). In Figure 6.34a we compare $\eta_{en,1}$, while in Figure 6.34b we analyze $\eta_{en,2}$. Neglecting head-tail hydrodynamic interactions greatly influences both the energetic efficiencies considered. Figures 6.34a and 6.34b show that, using the global approach, a maximum energetic efficiency emerges at intermediate flagellar lengths. This maximum cannot be detected using the additive approximation, while it is recovered using the v^2 correction.

Swimming Efficiency Following [70] a possible measure of the swimming efficiency is the linear distance covered per flagellar revolution, namely,

$$\eta_{sw,1G} = \frac{U}{\omega - \Omega}. \quad (6.89)$$

Using the additive approximation this becomes

$$\eta_{sw,1A} = \frac{B}{A + A_0}. \quad (6.90)$$

Alternatively the swimming efficiency can be defined as the translational velocity normalized by the motor torque, see [68],

$$\eta_{sw,2G} = \frac{U}{T_{motor}}. \quad (6.91)$$

In the additive approximation, this becomes

$$\eta_{sw,2A} = \frac{B}{C(A + A_0) - B^2}. \quad (6.92)$$

Since both (6.89) and (6.91) linearly depend on the swimming velocity, we use v to introduce the corrected swimming efficiencies as

$$\eta_{sw,1v} = v\eta_{sw,1A} = v\frac{B}{A + A_0}, \quad (6.93)$$

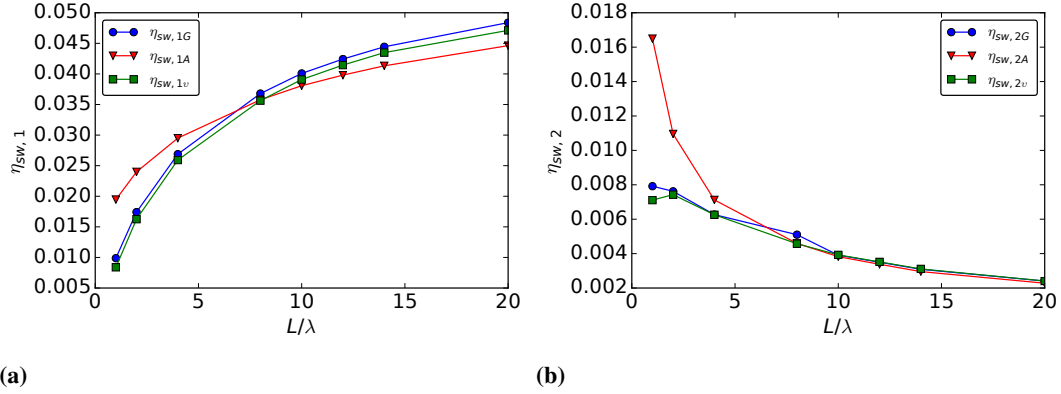


Figure 6.35: Analysis of the swimming efficiencies. Comparison between global approach (blue circles), local approximation (red triangles) and corrected local approach (green squares). In Figure 6.35a we show the analysis of the swimming efficiency $\eta_{sw,1}$. In Figure 6.35b we report the comparison of the swimming efficiency $\eta_{sw,2}$.

and

$$\eta_{sw,2v} = v\eta_{sw,2A} = v \frac{B}{C(A + A_0) - B^2}. \quad (6.94)$$

In Figure 6.35 we compare the swimming efficiencies computed using the additive approximation (red triangles), the global approach (blue circles) and the corrected additive approximation (green squares).

Neglecting the hydrodynamic interactions has an impact on the swimming efficiencies considered. In fact, the factor v guarantees much better accuracy of the prediction of the swimming efficiency computed using the additive approximation.

Conclusions and further developments

The results presented in this dissertation suggest how different Fluid Structure Interaction problems can be efficiently modelled through Boundary Element Methods. We developed efficient and accurate numerical tools that can be used both for engineering and scientific purposes.

In Chapter 1 we derived a BEM for the Laplace equation. We made use of high order elements, local refinement strategies, and complex geometries integration (via CAD data structures) to achieve efficient and accurate solutions of problems of industrial interest. In Chapter 2 we applied the BEM for the Laplace equation to solve linearized ship wave interaction problems. These results were achieved with an innovative BEM-FEM coupling resulting in a SUPG stabilized linearized free surface equation. Such an approach, described in Chapter 2, led to satisfying results both for submerged bodies and surface piercing hulls. We believe that such a condition could be an initial guess for a non linear stationary solver for the prediction of the wave drag of an advancing hull. Such a solver would close the gap between non linear unsteady solvers as [78, 81] and linearized ones [42].

The results of Chapters 3 and 4 proved the possibility of coupling different High Performance Computing libraries to reach an efficient, flexible and modular resolution of the dense linear systems naturally coming from the resolution of BEMs. In Chapter 3, we showed good performances, considering both strong and weak scalability, both for the standard BEM and the coupled BEM-FMA. We are currently studying ways to increase the number of degrees of freedom addressing the memory issues we have encountered when requiring more than 4×10^5 unknowns. Such an implementation would be a natural accelerator for the numerical methods described in Chapter 2 and in [81]. We applied the same parallelization techniques both to the Non Uniform Fast Fourier Transform [31, 47, 67] and to the Sparse Cardinal Sine Decomposition [3]. Although preliminary, the results showed that we developed a flexible modular NUFFT library that can be easily expanded by the users' community. Such a library is the keystone for the preliminary parallelization of the SCSD we proposed in Chapter 4. We are currently looking for an optimization of the preliminary parallelization results obtained with the Sparse Cardinal Sine Decomposition. Firstly we would like to compare the results obtained with the parallel BEM-FMA solver of Chapter 3 with the performances of the

Conclusions and further developments

parallel SCSD. Then, since SCSD has been successfully applied even to the Stokes system [4], we are currently performing a feasibility study regarding its implementation in the micro-motility framework.

In Chapter 5 and 6 we derived a parallel BEM for the Stokes system. We provided a complete procedure to retrieve affordable and accurate motility prevision both for “real-life” organism of experimental interest [50], and for “robotic-like” swimmer [91, 109]. The use of generic 3D modeling instruments allows us for a natural treatment of both eukaryotic and prokaryotic swimming strategies. The BEM solver we derived makes no assumption on the shape changes. We also used the solver to analyze a possible correction for data coming from different additive techniques as RFT or the approximation described in [96]. We proved that the additive approximation and, in particular, RFT miss completely the existence of optimal values of the flagellar length that maximize energetic efficiency, or swimming efficiency and lead to wrong predictions for the flagellar length giving maximal swimming speed U . The correcting factor v , for the predicted swimming speed U , is capable of improving the results that can be achieved using the additive approximation (or even RFT) for the swimming speed and for both kind of efficiencies, recovering the possibility of predicting optimal geometries which would be otherwise lost using the additive approximation.

A development of the present work lies in the introduction of a structural model for the flagella of swimmers. The BEM for the Stokes system could be coupled with such an elastic solver to complete the analysis of micro-motility. In such a case the shape changes would not be a prescribed Dirichlet datum but a consequence of the complete FSI problem. A natural input for the simulation would be the energy consumption of the swimmer, and it would be extremely interesting to study the optimal stroke in terms of such energy consumption.

Bibliography

- [1] Mark Ainsworth and J.Tinsley Oden. A posteriori error estimation in finite element analysis. *Computer Methods in Applied Mechanics and Engineering*, 142(1):1–88, Mar. 1997. DOI:10.1016/S0045-7825(96)01107-3.
- [2] J E Akin. *Finite Element Analysis with Error Estimators*. Elsevier, 2005.
- [3] François Alouges and Matthieu Aussal. The sparse cardinal sine decomposition and its application for fast numerical convolution. *Numerical Algorithms*, 70(2):427–448, Oct. 2015. DOI:10.1007/s11075-014-9953-6.
- [4] François Alouges, Matthieu Aussal, Aline Lefebvre-lepot, Franck Pigeonneau, and Antoine Sellier. The Sparse Cardinal Sine Decomposition applied to Stokes integral equations. Number 3, Florence, Italy, 2016. 9th International Conference on Multiphase Flow.
- [5] François Alouges, Antonio DeSimone, Laetitia Giraldi, and Marta Zoppello. Can Magnetic Multilayers Propel Artificial Microswimmers Mimicking Sperm Cells? *Soft Robotics*, 2(3):117–128, Sep. 2015. DOI:10.1089/soro.2015.0007.
- [6] Daniel Arndt, Wolfgang Bangerth, Denis Davydov, Timo Heister, Luca Heltai, Martin Kronbichler, Matthias Maier, Jean-Paul Pelteret, Bruno Turcksin, and David Wells. The deal.II Library, Version 8.5. *Journal of Numerical Mathematics*, 24(3), Jan. 2017. DOI:10.1515/jnma-2017-0058.
- [7] Marino Arroyo, Luca Heltai, D. Millan, and Antonio DeSimone. Reverse engineering the euglenoid movement. *Proceedings of the National Academy of Sciences*, 109(44):17874–17879, Oct. 2012. DOI:10.1073/pnas.1213977109.
- [8] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries. In *Modern Software Tools for Scientific Computing*, pages 163–202. Birkhäuser Boston, Boston, MA, 1997.

Bibliography

- [9] W Bangerth, C Burstedde, T Heister, and M Kronbichler. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Transaction on Mathematical Software*, 38:14/1—28, 2011. 2011.
- [10] W Bangerth, R Hartmann, and G Kanschat. deal.II - a General Purpose Object Oriented Finite Element Library. *ACM Transaction on Mathematical Software*, 33(4):24/1—24/27, 2007. 2007.
- [11] Wolfgang Bangerth, Jacky Austermann, B Markus, Juliane Dannberg, William Durkin, Thomas Geenen, Anne Glerum, Ryan Grove, Eric Heien, Martin Kronbichler, Elvira Mulyukova, Jonathan Perry-houts, Ian Rose, Cedric Thieulot, Iris Van Zelst, and Siqi Zhang. *ASPECT: Advanced Solver for Problems in Earth's ConvecTion*.
- [12] Yuanxun Bill Bao. A Parallel Implementation of the 3D NUFFT on Distributed-Memory Systems, 2015.
- [13] La Barba and Rio Yokota. How Will the Fast Multipole Method Fare in the Exascale Era? *SIAM News*, 46(6):8–9, 2013. DOI:10.1145/2160718.2160740.
- [14] Mario Bebendorf. *Hierarchical Matrices*, volume 63 of *Lecture Notes in Computational Science and Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
- [15] R Bebendorf, M. and Venn. No Title. *Journal of Numerical Mathematics*, 121:609–635, 2012. 2012.
- [16] K A Belibassakis, C Feurer, Th. P Gerostathis, A I Ginnis, P D Kaklis, J Kariagiannis, K V Kostas, D Mourkogiannis, C G Politis, and A Theodoulides. A BEM-Isogeometric Method with Application to the Wavemaking Resistance Problem of Ship at Constant Speed. In *ASME 2011 30th Conference on Ocean, Offshore and Arctic Engineering*, 2011.
- [17] Jose Luis Blanco. nanoflann: a c++ header-only fork of flann, a library for nearest neighbor (nn) with kd-trees. `\url{https://github.com/jlblancoc/nanoflann}`, 2014.
- [18] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2016.
- [19] C A Brebbia. *The Boundary Element Method for Engineers*. Pentech Press, 1978.
- [20] D. Brunner, M. Junge, P. Rapp, M. Bebendorf, and L. Gaul. Comparison of the fast multipole method with hierarchical matrices for the helmholtz-BEM. *CMES - Computer Modeling in Engineering and Sciences*, 58(2):131–158, 2010. 2010.
- [21] T Cao. Adaptive H- and H-R methods for Symm's integral equation. *Computer Methods in Applied Mechanics and Engineering*, 162(1-4):1–17, 1998. DOI:10.1016/S0045-7825(97)00326-5.

-
- [22] M. S. Celebi, M. H. Kim, and Robert F. Beck. Fully Nonlinear 3-D Numerical Wave Tank Simulation. *Journal of Ship Research*, 42(1):33–45, 1998. 1998.
 - [23] Jeng-tzong Chen, Chia-chun Hsiao, Yi-ping Chiu, and Ying-te Lee. Study of free-surface seepage problems using hypersingular equations. *Communications in Numerical Methods in Engineering*, 23(8):755–769, Oct. 2006. DOI:10.1002/cnm.925.
 - [24] XiaoBo Chen and Guo Xiong Wu. On singular and highly oscillatory properties of the Green function for ship motions. *Journal of Fluid Mech.*, 445:77–91, 2001. 2001.
 - [25] J Austin Cottrell, Thomas J R Hughes, and Yuri Bazilevs. *Isogeometric Analysis: Toward Integration of CAD and FEA*. Wiley Publishing, 1st edition, 2009.
 - [26] Frank S Crawford. Elementary derivation of the wake pattern of a boat. *American Journal of Physics*, 52:782–785, 1984. 1984.
 - [27] Baohu Dai, Jizhuang Wang, Ze Xiong, Xiaojun Zhan, Wei Dai, Chien-Cheng Li, Shien-Ping Feng, and Jinyao Tang. Programmable artificial phototactic microswimmer. *Nature Nanotechnology*, 11(12):1087–1092, 2016. DOI:10.1038/nnano.2016.187.
 - [28] Gianni Dal Maso, Antonio DeSimone, and Marco Morandotti. One-dimensional swimmers in viscous fluids: dynamics, controllability, and existence of optimal controls. *ESAIM: Control, Optimisation and Calculus of Variations*, 21(1):190–216, 2015. DOI:10.1051/cocv/2014023.
 - [29] C W Dawson. A practical computer method for solving ship-wave problems. In *Second International Conference on Numerical Ship Hydrodynamics*, pages 30–38, 1977.
 - [30] Alan Demlow and Gerhard Dziuk. An Adaptive Finite Element Method for the Laplace–Beltrami Operator on Implicitly Defined Surfaces. *SIAM Journal on Numerical Analysis*, 45(1):421–442, Jan. 2007. DOI:10.1137/050642873.
 - [31] Alok Dutt and V. Rokhlin. Fast Fourier Transforms for Nonequispaced Data. *SIAM Journal on Scientific Computing*, 14(6):1368–1393, 1993. DOI:10.1137/0914081.
 - [32] C Eck and W L Wendland. Numerische Mathematik A residual ϵ based error estimator for BEM ϵ discretizations of contact problems. *Numerische Mathematik*, pages 253–282, 2003. 2003.
 - [33] M. Frigo and S. G. Johnson. {The design and implementation of FFTW 3}. *Proceedings of the IEEE*, 93(2):216–231, 2005. DOI:10.1109/JPROC.2004.840301.
 - [34] S. Gaggero, D. Villa, and S. Brizzolara. RANS and PANEL method for unsteady flow propeller analysis. *Journal of Hydrodynamics*, 22(5 SUPPL. 1):547–552, 2010. DOI:10.1016/S1001-6058(09)60253-5.

Bibliography

- [35] Vipin Kumar George Karypis. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998. DOI:10.1137/S1064827595287997.
- [36] Veikko F Geyer, Frank Jülicher, Jonathon Howard, and Benjamin M Friedrich. Cell-body rocking is a dominant mechanism for flagellar synchronization in a swimming alga. *Proceedings of the National Academy of Sciences of the United States of America*, 110(45):18058–63, Nov. 2013. DOI:10.1073/pnas.1300895110.
- [37] Sharath S. Girimaji. Partially-Averaged Navier-Stokes Model for Turbulence: A Reynolds-Averaged Navier-Stokes to Direct Numerical Simulation Bridging Method. *Journal of Applied Mechanics*, 73(3):413, 2006. DOI:10.1115/1.2151207.
- [38] Nicola Giuliani. *An hybrid boundary element method for free surface flows*. PhD thesis, Politecnico di Milano, 2013.
- [39] Nicola Giuliani. BlackNUFFT, 2017.
- [40] Nicola Giuliani, Antonio Desimone, and Luca Heltai. Predicting and optimizing micro-swimmer performance from the hydrodynamics of its components: the relevance of interactions.
- [41] Nicola Giuliani, Andrea Mola, and Luca Heltai. pi-BEM: parallel BEM solver, 2017.
- [42] Nicola Giuliani, Andrea Mola, Luca Heltai, and Luca Formaggia. Engineering Analysis with Boundary Elements FEM SUPG stabilisation of mixed isoparametric BEMs : Application to linearised free surface flows. *Engineering Analysis with Boundary Elements*, 59:8–22, Oct. 2015. DOI:10.1016/j.enganabound.2015.04.006.
- [43] S.a. Goreinov, E.E. Tyrtyshnikov, and N.L. Zamarashkin. A theory of pseudoskeleton approximations. *Linear Algebra and its Applications*, 261(1-3):1–21, 1997. DOI:10.1016/S0024-3795(96)00301-1.
- [44] J. Gray and G. J. Hancock. The Propulsion of Sea-Urchin Spermatozoa. *Journal of Experimental Biology*, 32(4):802–814, 1955. 1955.
- [45] L. Greengard and W.D. Gropp. A parallel version of the fast multipole method. *Computers & Mathematics with Applications*, 20(7):63–71, 1990. DOI:10.1016/0898-1221(90)90349-O.
- [46] L Greengard and V Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348, 1987. DOI:10.1016/0021-9991(87)90140-9.
- [47] Leslie Greengard and June-Yub Lee. Accelerating the Nonuniform Fast Fourier Transform. *SIAM Review*, 46(3):443–454, 2004. DOI:10.1137/S003614450343200X.

-
- [48] S T Grilli and I A Svendsen. Corner problems and global accuracy in the boundary element solution of nonlinear wave flows. *Engineering Analysis with Boundary Elements*, pages 178–195, 1990. 1990.
 - [49] Stephan T Grilli, Philippe Guyenne, and Frédéric Dias. A fully non-linear model for three-dimensional overturning waves over an arbitrary bottom. *International Journal for Numerical Methods in Fluids*, 35:829–867, 2001. 2001.
 - [50] Jeffrey S. Guasto, Karl a. Johnson, and J. P. Gollub. Oscillatory Flows Induced by Microorganisms Swimming in Two Dimensions. *Physical Review Letters*, 105(16):168102, Oct. 2010. DOI:10.1103/PhysRevLett.105.168102.
 - [51] Morton Gurtin. *An Introduction to Continuum Mechanics (Mathematics in Science and Engineering)*. Academic Press, 1982.
 - [52] S. Han Aydin. FEM-BEM Coupling for the MHD Pipe Flow in an Exterior Region. *Open Journal of Fluid Dynamics*, 3(3):184–190, 2013. DOI:10.4236/ojfd.2013.33023.
 - [53] John Happel and Howard Brenner. *Low Reynolds number hydrodynamics*, volume 1 of *Mechanics of fluids and transport processes*. Springer Netherlands, Dordrecht, 1981.
 - [54] Christian Hasse, Volker Sohm, Martin Wetzel, and Bodo Durst. Hybrid URANS/LES turbulence simulation of vortex shedding behind a triangular flameholder. *Flow, Turbulence and Combustion*, 83(1):1–20, 2009. DOI:10.1007/s10494-008-9186-7.
 - [55] T H Havelock. The Wave Resistance of a Spheroid. *Proceedings of the Royal Society*, pages 275–285, 1931. 1931.
 - [56] Luca Heltai, Marino Arroyo, and Antonio DeSimone. Nonsingular isogeometric boundary element method for Stokes flows in 3D. *Computer Methods in Applied Mechanics and Engineering*, 268:514–539, 2014. DOI:10.1016/j.cma.2013.09.017.
 - [57] Michael a. Heroux, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, Kendall S. Stanley, Roscoe a. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, and Roger P. Pawlowski. An overview of the Trilinos project. *ACM Transactions on Mathematical Software*, 31(3):397–423, 2005. DOI:10.1145/1089014.1089021.
 - [58] R Hiptmair and L Kielhorn. BETL – A generic boundary element template library. *Seminar for Applied Mathematics, ETH Zürich*, 36, 2012. 2012.
 - [59] Christine Josenhans and Sebastian Suerbaum. Motility in bacteria. *International Journal of Medical Microbiology*, 291:605–614, 2002. 2002.
 - [60] Jang Jou and Jinn-Liang Liu. A posteriori boundary element error estimation. *Journal of Computational and Applied Mathematics*, 106(1):1–19, 1999. DOI:10.1016/S0377-0427(99)00049-7.

Bibliography

- [61] Dhiraj D. Kalamkar, Joshua D. Trzaskoz, Srinivas Sridharan, Mikhail Smelyanskiy, Daehyun Kim, Armando Manduca, Yunhong Shu, Matt a. Bernstein, Bharat Kaul, and Pradeep Dubey. High Performance Non-uniform FFT on Modern X86-based Multi-core Systems. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pages 449–460. IEEE, May. 2012.
- [62] Philipp Kanehl and Takuji Ishikawa. Fluid mechanics of swimming bacteria with multiple flagella. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 89(4):1–8, 2014. DOI:10.1103/PhysRevE.89.042704.
- [63] Martin Kronbichler, Timo Heister, and Wolfgang Bangerth. High accuracy mantle convection simulation through modern numerical methods. *Geophysical Journal International*, 191(1):12–29, 2012. DOI:10.1111/j.1365-246X.2012.05609.x.
- [64] A. G. Lamorgese, D. A. Caughey, and S. B. Pope. Direct numerical simulation of homogeneous turbulence with hyperviscosity. *Physics of Fluids*, 17(1):1–11, 2005. DOI:10.1063/1.1833415.
- [65] Ilya Lashuk, George Biros, Aparna Chandramowliswaran, Harper Langston, Tuan-Anh Nguyen, Rahul Sampath, Aashay Shringarpure, Richard Vuduc, Lexing Ying, and Denis Zorin. A massively parallel adaptive fast multipole method on heterogeneous architectures. *Ieee-Micro*, 55(5):101, 2012. DOI:10.1145/2160718.2160740.
- [66] Eric Lauga, Willow R DiLuzio, George M Whitesides, and Howard A Stone. Swimming in circles: motion of bacteria near solid boundaries. *Biophysical Journal*, 90(2):400–12, 2006. DOI:10.1529/biophysj.105.069401.
- [67] June Yub Lee and Leslie Greengard. The type 3 nonuniform FFT and its applications. *Journal of Computational Physics*, 206(1):1–5, 2005. DOI:10.1016/j.jcp.2004.12.004.
- [68] Guanglei Li and Jay X Tang. Low flagellar motor torque and high swimming efficiency of *Caulobacter crescentus* swarmer cells. *Biophysical Journal*, 91(7):2726–34, 2006. DOI:10.1529/biophysj.106.080697.
- [69] James Lighthill. Flagellar Hydrodynamics. *SIAM Review*, 18(2):161–230, Apr. 1976. DOI:10.1137/1018040.
- [70] Yukio Magariyama and Seishi Kudo. A mathematical explanation of an increase in bacterial swimming speed with viscosity in linear-polymer solutions. *Biophysical Journal*, 83(2):733–739, 2002. DOI:10.1016/S0006-3495(02)75204-1.
- [71] Dhairya Malhotra and George Biros. Algorithm 967: A Distributed-Memory Fast Multipole Method for Volume Potentials. *ACM Transaction on Mathematical Software*, 43(2):17:1—17:27, Aug. 2016. DOI:10.1145/2898349.
- [72] Andrea Manzoni, Filippo Salmoiraghi, and Luca Heltai. Reduced Basis Isogeometric Methods (RB-IGA) for the real-time simulation of potential flows about

- parametrized NACA airfoils. *Computer Methods in Applied Mechanics and Engineering*, 284:1147–1180, 2015. DOI:10.1016/j.cma.2014.11.037.
- [73] Gianni Dal Maso, Antonio DeSimone, and Marco Morandotti. An Existence and Uniqueness Result for the Motion of Self-Propelled Microswimmers. *SIAM Journal on Mathematical Analysis*, 43(3):1345–1368, Jan. 2011. DOI:10.1137/10080083x.
- [74] M.Ikehata, H.Tanaka, H.Adachi, M.Namimatsu, and S.Ogiwara. The summary of the cooperative experiment on Wigley parabolic model in Japan, 1983.
- [75] Parviz Moin and Krishnan Mahesh. DIRECT NUMERICAL SIMULATION: A Tool in Turbulence Research. *Annual Review of Fluid Mechanics*, 30(1):539–578, 1998. DOI:10.1146/annurev.fluid.30.1.539.
- [76] Andrea Mola. *Model for Olympic Rowing Boats*. PhD thesis, Politecnico di Milano, 2009.
- [77] Andrea Mola, Luca Heltai, and Antonio Desimone. A stable semi-lagrangian potential method for the simulation of ship interaction with unsteady and nonlinear waves. In *17th International Conference on Ships and Shipping Research*, 2012.
- [78] Andrea Mola, Luca Heltai, and Antonio DeSimone. A stable and adaptive semi-Lagrangian potential model for unsteady and nonlinear ship-wave interactions. *Engineering Analysis with Boundary Elements*, 37(1):128–143, Jan. 2013. DOI:10.1016/j.enganabound.2012.09.005.
- [79] Andrea Mola, Luca Heltai, and Antonio Desimone. A fully nonlinear potential model for ship hydrodynamics directly interfaced with CAD data structures. In *The 24th International Ocean and Polar Engineering Conference*, 2014.
- [80] Andrea Mola, Luca Heltai, and Antonio Desimone. Nonlinear free surface potential flow simulations for hulls with a transom stern operating in dry and wet conditions. In *18th International Conference on Ships and Shipping Research*, 2015.
- [81] Andrea Mola, Luca Heltai, and Antonio DeSimone. Wet and Dry Transom Stern Treatment for Unsteady and Nonlinear Potential Flow Model for Naval Hydrodynamics Simulations. *Journal of Ship Research*, 61(1):1–14, Mar. 2017. DOI:10.5957/JOSR.61.1.160016.
- [82] Luigi Morino. Subsonic Potential Aerodynamics for Complex Configurations : A General Theory. *AIAA Journal*, 12(2):191–197, Feb. 1974. DOI:10.2514/3.49191.
- [83] S Nam and Ta Basha. A GPU implementation of compressed sensing reconstruction of 3D radial (kooshball) acquisition for high-resolution cardiac MRI. *Proceeding International Society Magnetic Resonance Medicine*, 19:730339, 2011. 2011.

Bibliography

- [84] J N Newman and J M Clarisse. Evaluation of the wave-resistance Green function near the singular axis. *J Ship Research*, 38, 2004. 2004.
- [85] J N Newmann. *Marine Hydrodynamics*. the MIT Press, 1977.
- [86] Francis Noblesse, Fuxin Huang, and Chi Yang. The Neumann-Michell theory of ship waves . *J Eng Math*, 2012. 2012.
- [87] G. Of. An efficient algebraic multigrid preconditioner for a fast multipole boundary element method. *Computing*, 82(2-3):139–155, 2008. DOI:10.1007/s00607-008-0002-y.
- [88] E. Passov and Y. Or. Dynamics of Purcell’s three-link microswimmer with a passive elastic tail. *European Physical Journal E*, 35(8), 2012. DOI:10.1140/epje/i2012-12078-9.
- [89] Dmitry Pekurovsky. P3DFFT: A Framework for Parallel Computations of Fourier Transforms in Three Dimensions. *SIAM Journal on Scientific Computing*, 34(4):C192–C209, Jan. 2012. DOI:10.1137/11082748X.
- [90] N. Phan-Thien, T. Tran-Cong, and M. Ramia. A boundary-element analysis of flagellar propulsion. *Journal of Fluid Mechanics*, 184(-1):533, Nov. 1987. DOI:10.1017/S0022112087003008.
- [91] D Pimponi, M Chinappi, P Gualtieri, and C M Casciola. Hydrodynamics of flagellated microswimmers near free-slip interfaces. *Journal of Fluid Mechanics*, 789:514–533, 2016. DOI:10.1017/jfm.2015.738.
- [92] Michael Pippig and Daniel Potts. Parallel Three-Dimensional Nonequispaced Fast Fourier Transforms and Their Application to Particle Simulation. *SIAM Journal on Scientific Computing*, 35(4):C411–C437, 2013. DOI:10.1137/120888478.
- [93] C. Pozrikidis. *Boundary Integral and Singularity Methods for Linearized Viscous Flow*, volume 36. Cambridge University Press, Cambridge, 1992.
- [94] Jacek Ptaszny. Accuracy of the fast multipole boundary element method with quadratic elements in the analysis of 3D porous structures. *Computational Mechanics*, 56(3):24–28, 2015. DOI:10.1007/s00466-015-1182-x.
- [95] E. M. Purcell. Life at low Reynolds number. *American Journal of Physics*, 45(1):3–11, 1977. DOI:10.1119/1.10903.
- [96] Edward M Purcell. The efficiency of propulsion by a rotating flagellum. *Biophysics*, 94(October):11307–11311, 1997. DOI:10.1073/pnas.94.21.11307.
- [97] M Ramia, D L Tullock, and N Phan-Thien. The Role of Hydrodynamic Interaction in the Locomotion of Microorganisms. *Biophysical Journal Volume*, 65(August):755–778, 1993. DOI:10.1016/S0006-3495(93)81129-9.
- [98] Hoyte C Raven. *A Solution Method for the Nonlinear Ship Wave Resistance Problem*. PhD thesis, Technische Universiteit Delft, 1996.

-
- [99] James Reinders. *Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism*. O'Reilly & Associates, Inc., first edition, 2007.
 - [100] Bruce Rodenborn, Chih-Hung Chen, Harry L. Swinney, Bin Liu, and H. P. Zhang. Propulsion of microorganisms by a helical flagellum. *Proceedings of the National Academy of Sciences of the United States of America*, 110(5):E338–E347, 2013. DOI:10.1073/pnas.1219831110.
 - [101] Youcef Saad and Martin H. Schultz. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986. DOI:10.1137/0907058.
 - [102] Alberto Sartori, Nicola Giuliani, Mauro Bardelloni, and Luca Heltai. deal2lkit : a Toolkit Library for High Performance Programming in deal . II. *submitted*, pages 1–26, 2015. 2015.
 - [103] Alberto Sartori, Nicola Giuliani, Mauro Bardelloni, and Luca Heltai. deal2lkit: a Toolkit Library for deal.II. Technical Report 57/2015/MATE, SISSA, 2015.
 - [104] Johannes Schindelin, Curtis T Rueden, Mark C Hiner, and Kevin W Elieci. The ImageJ ecosystem: An open platform for biomedical image analysis. *Molecular Reproduction and Development*, 82(7-8):518–529, 2015. DOI:10.1002/mrd.22489.
 - [105] Thomas Schiwietz, Ti-chiun Chang, Peter Speier, and Rüdiger Westermann. MR Image Reconstruction Using the GPU. *SPIE Medical Imaging*, pages 1279–1290, 2006. DOI:10.1117/12.652223.
 - [106] S M Scorpio. *Fully nonlinear ship-wave computations using a multipole accelerated, desingularized method*. Report (University of Michigan. Dept. of Naval Architecture and Marine Engineering). University of Michigan, 1997.
 - [107] David C Scullen. *Accurate Computation of Steady Nonlinear Free-Surface Flows*. PhD thesis, University of Adelaide, 1998.
 - [108] Yunhong Shu, Matt A. Bernstein, John Huston, and Dan Rettmann. Contrast-enhanced intracranial magnetic resonance angiography with a spherical shells trajectory and online gridding reconstruction. *Journal of Magnetic Resonance Imaging*, 30(5):1101–1109, 2009. DOI:10.1002/jmri.21938.
 - [109] H. Shum, E. a. Gaffney, and D. J. Smith. Modelling bacterial behaviour close to a no-slip plane boundary: the influence of bacterial geometry. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 466(2118):1725–1748, 2010. DOI:10.1098/rspa.2009.0520.
 - [110] Henry Shum and Eamonn A Gaffney. Hydrodynamic analysis of flagellated bacteria swimming near one and between two no-slip plane boundaries. 033012:1–10, 2015. DOI:10.1103/PhysRevE.91.033012.
 - [111] Wojciech Śmigaj, Timo Betcke, Simon Arridge, Joel Phillips, and Martin Schweiger. Solving Boundary Integral Problems with BEM++. *ACM Transactions on Mathematical Software*, 41(2):1–40, 2015. DOI:10.1145/2590830.

Bibliography

- [112] P R Spalart, S R Allmaras, and January Reno. A One-Equation Turbulence Model for Aerodynamic Flows Boeing Commercial Airplane Group 30th Aerospace Sciences. *AIAA paper 1992-0439*, 1992. DOI:10.2514/6.1992-439.
- [113] Olaf Steinbach. *Numerical Approximation Methods for Elliptic Boundary Value Problems*. Springer New York, New York, NY, 2008.
- [114] J C F Telles. A Self-Adaptive Co-ordinate Transformation For Efficient Numerical Evaluation of General Boundary Element Integrals. *International Journal for Numerical Methods in Engineering*, 24:959–973, 1987. 1987.
- [115] Bruno Turcksin, Martin Kronbichler, and Wolfgang Bangerth. Work-Stream – A Design Pattern for Multicore-Enabled Finite Element Computations. *ACM Transactions on Mathematical Software*, 43(1):1–29, Aug. 2016. DOI:10.1145/2851488.
- [116] X Wang, C Liu, Z Sun, M Wu, and S Zhang. Inherent Properties of Two Dimension Green Function with Linear Boundary Condition of Free Water Surface. *Applied Mathematics*, 4:97–99, 2013. 2013.
- [117] Zhihua Wang, Pei He, Yu Lv, Junhu Zhou, Jianren Fan, and Kefa Cen. Direct numerical simulation of subsonic round turbulent jet. *Flow, Turbulence and Combustion*, 84(4):669–686, 2010. DOI:10.1007/s10494-010-9248-5.
- [118] DAVID C. WILCOX. Reassessment of the scale-determining equation for advanced turbulence models. *AIAA Journal*, 26(11):1299–1310, 1988. DOI:10.2514/3.10041.
- [119] Xue-Hong Wu, Zhi-Juan Chang, Yan-Li Lu, Wen-Quan Tao, and Sheng-Ping Shen. An analysis of the convection–diffusion problems using meshless and meshbased methods. *Engineering Analysis with Boundary Elements*, 36:1040, 2012. 2012.
- [120] Ronal W Yeung. Numerical methods in free-surface flows. *Annual Review of Fluid Mechanics*, pages 395–442, 1982. 1982.
- [121] Rio Yokota and Lorena Barba. A Tuned and Scalable Fast Multipole Method as a Preeminent Algorithm for Exascale Systems. 2011. DOI:10.1177/1094342011429952.
- [122] Rio Yokota, Jaydeep P. Bardhan, Matthew G. Knepley, L. a. Barba, and Tsuyoshi Hamada. Biomolecular electrostatics using a fast multipole BEM on up to 512 gpus and a billion unknowns. *Computer Physics Communications*, 182(6):1272–1283, 2011. DOI:10.1016/j.cpc.2011.02.013.